



**Konzept und Ansatz einer Wertschöpfungskette für die Erkennung und
Bereitstellung neuer Fahrumfänge intelligenter Fahrzeuge**
Bachelorarbeit

An der
Carl von Ossietzky Universität Oldenburg
Studiengang Wirtschaftsinformatik

Vorgelegt von Linus Hestermeyer
linus.hestermeyer@gmail.com
Matr.Nr.: 4087097

Erstprüfer: Prof. Dr. Frank Köster
Zweitprüfer: Dipl.-Inform. Gerald Sauter

Oldenburg, den 14. Juni 2020

Vorwort

Diese Arbeit entstand im Rahmen meines Wirtschaftsinformatik Studiums an der Carl von Ossietzky Universität Oldenburg.

Mein Besonderer Dank gilt meinen Betreuern Gerald Sauter und Prof. Dr. Frank Köster für die stetige fachliche Betreuung und Hilfe bei der Erstellung der Arbeit und des Prototypen. Der Gedankenaustausch über die Thematik und die Änderung des Kernthemas der Arbeit haben das Ergebnis erst möglich gemacht.

Selbstverständlich gilt mein Dank auch meinen Eltern und Geschwistern Lukas und Johanna, die mich bei der Entwicklung meiner fachlichen und beruflichen Laufbahn stets unterstützt haben.

Nicht zuletzt möchte ich meiner Freundin Julia, der Fachschaft Informatik der Universität Oldenburg und dem OFFIS danken, welche während des Studiums immer eine gute Anlaufstelle für fachliche aber auch persönliche Aspekte meiner selbst gewesen sind.

Oldenburg, im Juni 2020

Linus Hestermeyer

Inhaltsverzeichnis

1 Motivation	1
2 Theoretischer Rahmen	2
2.1 Einführung	2
2.2 Bedarfserkennung von Software	3
2.2.1 Relevante Systeme eines autonomen Fahrzeugs in der Bedarfserkennung	3
2.2.2 Eine geeignete Kommunikationsgrundlage: OpenSCENARIO	4
2.2.3 Software-User-Pattern-Recognizer [SUPR] im Rahmen der Bedarfserkennung	6
2.3 Bereitstellung von Software	8
2.3.1 OTA-Aktualisierungen mit UPTANE	8
2.3.2 Anpassung von UPTANE Architektur	9
2.3.3 SUPR im Rahmen der Softwarebereitstellung	11
2.3.4 Die Mensch-Maschine-Schnittstelle	15
2.4 Technologie- und Methodik-Scouting	16
2.4.1 Arbeitsmethodik: User-Centered-Design	16
2.4.2 Tech-Scouting	19
2.5 Ausblick und erstes Konzept der praktischen Umsetzung	20
3 Das Business Model Canvas und Wertschöpfungskette	24
3.1 Business Model Canvas	24
3.1.1 Kundensegmente	25
3.1.2 Kundenbeziehungen	26
3.1.3 Marketingkanäle	26
3.1.4 Nutzenversprechen	26
3.1.5 Einnahmequellen	27
3.1.6 Schlüsselressourcen	28
3.1.7 Schlüsselaktivitäten	28
3.1.8 Schlüsselpartner	29
3.1.9 Kostenstruktur	30
3.2 Die Supply Chain und die Wertschöpfungskette	31
3.2.1 Primäre Aktivitäten	31
3.2.2 Unterstützende Aktivitäten	36
3.3 Zusammenfassung	38
4 Technische Konzepte	39
4.1 Klassifizierung von Software	39
4.1.1 Berechtigungen	39
4.1.2 Abgeleitete Kennzahlen	40
4.2 Umgebungsbedingte Softwaresuche	41
4.3 Kommunikationsprotokolle	42
4.3.1 Kommunikationsprotokoll: Selbstständiger Softwarekauf	42

4.3.2	Kommunikationsprotokoll: Installationsvorschlag vom Service Provider	43
4.3.3	Kommunikationsprotokoll: Nutzung eines Service	43
4.4	Ausblick	46
5	Vorstellung des Prototypen	46
5.1	Architektur des Prototypen	46
5.1.1	Architektur des Software Applikation Managers (SAM)	48
5.1.2	Architektur des OEM-Servers (Server)	49
5.2	Installation	49
5.2.1	Server und SAM Installation	50
5.2.2	MMS Installation	50
5.2.3	Carla Installation	51
5.3	Nutzung des Prototypen	52
5.4	Analyse des Prototypen und Ausblick auf die Weiterentwicklung	53
6	Reflexion der Ergebnisse	55
7	Ausblick	56

Abbildungsverzeichnis

1	Architektur AV nach Jeff Schneider (CMU) [5]	4
2	Ausschnitt einer OpenSENARIO Datei von Andreas Biehn [8, (Downloads)]	5
3	Umweltanalyse	7
4	Architektur Design UPTANE [7]	8
5	Anangepasste Uptane Architektur	10
6	Technisches Erstkonzept	22
7	Supply Chain von Software	31
8	Die wichtigsten Bausteine einzelner Aktivitäten der Wertschöpfungskette	32
9	Jeff Schneider: Architecture of Autonomous Vehicles	39
10	Eigenständige Installation von Software	42
11	Installationsprozess von Software	44
12	Nutzungsprozess von Software	45
13	Grundlegende Architektur der Prototypen	47
14	Aufbau der Common Library	48
15	Architektur des Software Applikation Managers	48
16	Architektur des Software Applikation Managers	49

1 Motivation

Das autonome Fahren wird als die Zukunft des Automobils gesehen. Wann Fahrzeuge jedoch tatsächlich vollständig fahrerlos Fahren ist noch nicht absehbar. Künftig entwickelte Softwares können Fahrzeugen einzelne autonome Fahrfunktionen hinzufügen, durch welche diese zunehmend mehr selbstständig fahren können. Damit diese Softwares auch nach dem Kauf des Fahrzeuges selbstständig installiert werden können, soll der Kauf und die Installation dieser über eine kabellose Schnittstelle ermöglicht werden. Dies kann durch die Integration einer Shop-Plattform in das Fahrzeug realisiert werden, über welche Fahrzeughalter eigenständig auswählen können welche Softwares auf dem Fahrzeug installiert werden sollen. Um dies verwirklichen zu können muss festgestellt werden, *was relevante Bausteine einer Wertschöpfungskette für die Erkennung und Bereitstellung neuer Fahrumfänge für intelligente Fahrzeuge sein können und wie Automobilherstellern diese umsetzen könnten*. Es wird ein Prototyp erstellt, welcher den Kauf, die Installation und Nutzung einer Software darstellt.

In Kapitel 3 wird anhand eines Business Model Canvas ein Überblick des Marktes für Fahrunktionssoftware geschaffen. Die dort identifizierten Schlüsselaktivitäten werden daraufhin erläutert und in eine Wertschöpfungskette eingearbeitet. In Kapitel 4 werden technischen Konzepte vorgestellt, die einzelne Bausteine der Wertschöpfungskette zusätzlich detaillieren. Es wird ein Konzept zur Klassifizierung von Software vorgestellt, nach der Art der unterschiedlichen Zugriffsrechte auf die Systeme des Fahrzeugs. Für die individuelle Bedarfsbestimmung von Software wird ein Suchalgorithmus skizziert und es werden Kommunikationsprotokolle erstellt, welche die Interaktion zwischen dem Server, einem Fahrzeug und Serviceprovider darstellen. In Kapitel 5 wird der Prototyp vorgestellt, welcher die Ergebnisse der vorigen Kapitel zusammenfassen und visuell veranschaulichen soll. Hierzu wird zunächst die Architektur des Prototypen vorgestellt und verdeutlicht, an welchen Stellen die jeweiligen Konzepte integriert wurden.

Zunächst werden hierzu bestehende Technologien und eigens erstellte System-Konzepte vorgestellt, welche die Bedarfserkennung und Bereitstellung von Software unterstützen.

Das folgende Kapitel während des vorangegangenen Forschungsseminars erstellt.

2 Theoretischer Rahmen

2.1 Einführung

"Durch den Einsatz von Elektrik und Elektronik ist das Automobil in den vergangenen Jahrzehnten stark geprägt worden, und die „Intelligenz“ in den Subsystemen hat exponentiell zugenommen."(Vgl. [4, S. 1]) Mit zunehmend mehr intelligenten Fahrassistenten in modernen Fahrzeugen ist es Absehbar, dass bereits in naher Zukunft Fahrzeuge die Stufe 3 bzw. teilweise Stufe 4 des Autonomen Fahrens erreichen. Ein Fahrzeug der Stufe 3 ist dazu in der Lage dazu, die Längs- und Querlenkung in bestimmten Anwendungsfällen selber übernehmen und diese so sicher zu durchfahren. Hierbei wäre allerdings noch ein Insasse notwendig, der in einem Notfall das Steuer übernehmen kann. In Stufe 4 Fahrzeugen kann das Fahrzeug die komplette Fahraufgabe in bestimmten Anwendungsfällen übernehmen. [17, S. 14]

Erst Stufe 5 Fahrzeuge werden "volumänglich auf allen Straßentypen, in allen Geschwindigkeitsbereichen und unter allen Umfeldbedingungen die Fahraufgabe vollständig allein durchführen. Wann dieser Automatisierungsgrad erreicht sein wird, kann heute noch nicht benannt werden."(Vgl. [17, S. 14]). Aufgrund dessen ist es wichtig, dass Stufe 3 und Stufe 4 Fahrzeuge in Zukunft mehr Anwendungsfälle abdecken können. Hierzu sollen diese über eine kabellose Schnittstelle orts- und zeitunabhängig Softwarepakete herunterladen können, welche das Spektrum der autonomem Fahrfunktionen des Fahrzeugs zweckgebunden erweitert. Angenommen Sie planen mit ihrem neuen Fahrzeug eine Autofahrt nach England. Spätestens ab dem Ende des Eurotunnels wäre es für das Fahrzeug nicht mehr möglich selbstständig zu fahren, da dort Linksverkehr herrscht. Das Auto soll automatisch erkennen können, dass es ab einem bestimmten Punkt nicht mehr selbstständig fahren kann. In Folge dessen soll es eine Software zum Kauf/Miete anbieten, welche es dem Auto ermöglicht, am Linksverkehr teilnehmen zu können. Die Halter können ihr Fahrzeug dementsprechend zunehmend autonom fahren lassen, was den Marktwert des Autos nach dem Kauf steigern kann. Die Fahrzeughalter sollen bei Kauf von Software vom System unterstützt werden in Form von Vorschlägen für neue Software, die den Bedarf des Fahrers abdeckt. Diese Vorschläge sollen zu Zeitpunkten erfolgen, in denen der Verkauf einer bestimmten Software möglichst wahrscheinlich ist. Mittels dessen wird zudem unterbunden, dass der Nutzer von zu vielen Benachrichtigungen überfordert wird.

Damit Anwendungsfälle rasch abgedeckt werden können, bedarf es einem großen Spektrum an Software, die eben diese abdeckt. Um dies schnellstmöglich bewerkstelligen zu können ist es erforderlich, dass die Entwicklung von Softwarepakete durch Zulieferer geschehen, welche sich explizit auf diesen Markt fokussieren. Der hierdurch entstehende Seitenmarkt für die Entwicklung von Fahrfunktionssoftware kann somit schnell wachsen und sich als Teil in der Automobilindustrie etablieren. Durch autonome Fahrfunktionen werden Autos in der Regel schonender gefahren als vom Menschen, weshalb die Lebenszeit von Autos voraussichtlich verlängert wird. [17, (S. 16)] Ericssons Juergen Daunis sagt diesbezüglich, dass die meisten Analytiker und Führungskräfte der Meinung sind, dass der Umsatz dieser neu entstehenden Seitenmärkte weiter steigen wird und dass das traditionelle Geschäftsmodell an dem wirtschaftlichen Maximum seiner Existenz ist. [6]

Die in dieser Arbeit erfassten Gliederungspunkte dienen als Grundlage für die gleichnamige Bachelorarbeit und sind im Wesentlichen als Literaturrecherche und Grundlagenerarbeitung zu verstehen. Es werden einzelne Bausteine der in der Bachelorarbeit zu erstellenden Wertschöpfungskette erstmalig benannt und konkretisiert. In Kapitel zwei wird dargestellt, wie das Auto einen Softwarebedarf erkennt, wie es einen Server hierüber informiert und wie dieser Server die richtige Software sucht. In Kapitel Drei wird zum einen eine sichere Architektur für kabellose Aktualisierungen erarbeitet. Des weiteren wird erläutert, wie Software verkauft wird und es werden Richtlinien für die Mensch-Maschine-Schnittstelle erarbeitet, auf welcher ein Angebot letzten Endes angezeigt werden soll.

2.2 Bedarfserkennung von Software

Im Rahmen der Bedarfserkennung steht die Beschreibung der eigenen Umgebung von Fahrzeugen im Mittelpunkt. Damit die Softwarehersteller den Bedarf von Fahrzeugen abdecken können, muss ihnen die Umgebung des Fahrzeugs zu dem Zeitpunkt der Bedarfserkennung bekannt sein. Um zu verstehen, wie ein Fahrzeug seine Umwelt eigens beschreiben kann, wird im Folgenden die Architektur eines autonomen Fahrzeugs vorgestellt und erläutert, welche Bestandteile dessen für die Bedarfserkennung relevant sind.

2.2.1 Relevante Systeme eines autonomen Fahrzeugs in der Bedarfserkennung

Um die Suche nach Software zu erleichtern, benötigt die Suche auf dem Server gewisse Daten als Input. Damit die Situation, in welcher das Fahrzeug nicht selbstständig fahren kann, durch eine Software abgedeckt werden kann ist es wichtig eine Beschreibung der Umwelt des Autos zu dem Zeitpunkt der Bedarfserkennung zu erstellen. Durch die Fusion aufgenommener Kamera-, Ultraschall- und Radardaten kann die Umwelt beziehungsweise die Umgebung des Autos in einem Datenformat wie dem von der ASAM.¹ definierten Standard "OpenSCENARIO" [8] dargestellt werden. Dieser wird in Kapitel 2.2.2 vorgestellt.

Im Rahmen der Suche nach möglicherweise nötiger Software spielt die Routen- und Bewegungsplanung des Autos eine große Rolle. Zum einen kann das Auto bei der Eingabe einer neuen Route diese nach Gegenden absuchen in denen oft neue Software benötigt wird. Es kann infolgedessen diese dem Fahrer bereits vor Fahrtbeginn vorschlagen und somit die Bequemlichkeit der Fahrt sicherstellen. Zum anderen soll das Auto Muster im Fahrtenverlauf erkennen, um so bei der Erkennung eines Softwarebedarfs auf einer dieser Strecken eben diese Software zum Kauf vorzuschlagen.

Abbildung 1 zeigt die Architektur eines Autonomen Fahrzeugs nach Jeff Schneider. Sie verdeutlicht, welche Bestandteile ein Autonomes Fahrzeug besitzt um mit Hilfe dieser selbstständig zu fahren. Die aufgenommenen *Sensordaten* leiten Informationen an die *Karten- & Positionsverfolgung* weiter. Durch einen Merge (Zusammenführung) dieser Daten kann das Fahrzeug die eigene Umwelt *wahrnehmen* und mit Hilfe der dynamischen Inhalte der Welt (zB. Geschwindigkeiten) *Vorhersagen* für den Verkehr stellen. Als Bündel stellen sie die Bewegungsplanung dar.

¹<https://www.asam.net/standards/detail/openscenario/>

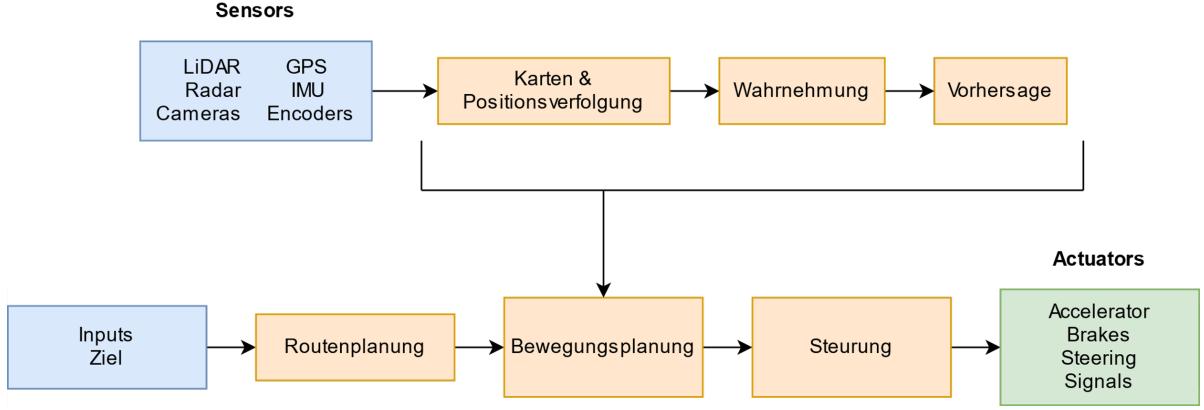


Abbildung 1: Architektur AV nach Jeff Schneider (CMU) [5]

Der Fahrer gibt dem Auto initial ein *Ziel*, mit Hilfe dessen das Fahrzeug die zu fahrende Route berechnet. Die Bewegungssteuerung gibt das Wahrgenommene und Vorhergesagte weiter an die Steuerung welche letztendlich entscheidet, was die einzelnen Aktuatoren machen. Damit der Bedarf einer Softwarelücke festgestellt werden kann, müssen die Systembausteine "Sensoren", "Karten & Positionsverfolgung", "Wahrnehmung" und "Vorhersage" angeknüpft werden um so herauszufinden **wann** das Fahrzeug nicht mehr selbstständig fahren kann. Wurde dieser Moment festgestellt, wird die Übergabe der Fahraufgabe initiiert und zeitgleich auch die Suche nach Software gestartet.

Neben den wichtigen Bausteinen der Bewegungsplanung kann auch anhand der Routenplanung von Fahrzeugen ein Softwarebedarf untersucht werden (Siehe 2.2.3). Zunächst wird eine Möglichkeit dargestellt, wie die Daten der Bewegungsplanung in einer geeigneten Form gespeichert und versendet werden können.

2.2.2 Eine geeignete Kommunikationsgrundlage: OpenSCENARIO

Der Inhalt dieses Kapitels basiert auf Inhalten der Projektwebseiten des OpenSCENARIO Standards. [8]. OpenSCENARIO ist ein XML-basiertes Dateiformat zur Beschreibung aller statischen (*Gegenstände, Hindernisse, etc.*) und dynamischen (*Geschwindigkeiten, Bewegungsrichtungen, etc.*) Inhalte der Umwelt eines Fahrzeugs. "Der primäre Anwendungsfall von OpenSCENARIO ist es komplexe, synchrone Manöver zu beschreiben, welche mehrere Entitäten wie Fahrzeuge, Fußgänger und andere Verkehrsteilnehmer betreffen." (Vgl. [8, asam.net])

Abbildung 2 zeigt einen Ausschnitt einer OpenSCENARIO-Datei. Für ein Szenario ist immer das **Strassenetzwerk** (*RoadNetwork*) sowie die beinhalteten **Entitäten** (*Entities*) festzulegen. Das eigentliche Szenario ist innerhalb eines **Storyboards** dargestellt und unterteilt sich in einzelne **Storys** (Siehe Abbildung 2). Alle zuvor definierten Entitäten erhalten eine initiale **Action**, welche das initiale Handeln der Entität festlegt (Siehe <Init>-Block).

Eine einzelne Story ist immer einer einzelnen Entität zuzuordnen. Eine Story wiederum ist in **Acts** aufgeteilt, welcher eine Sammlung an **Sequenzes** beinhalten kann. Jedes dieser Elemente kann mit einer **Condition** versehen werden. Wird die "Condition" (Bedingung) erfüllt, wird der jeweilige Story-/Act- oder Sequenz-Block ausgeführt.

```

1   <?xml version="1.0" encoding="utf-8"?>
2   <OpenSCENARIO>
3
4   <FileHeader revMajor="0" revMinor="9" date="2017-02-24T10:00:00"
5     description="Sample Scenario - Overtaker" author="Andreas Biehn"/>
6
7   <ParameterDeclaration/>
8
9   <Catalogs>
38
39   <RoadNetwork>
40     <Logics filepath="Databases/PEGASUS/PEGASUS_A01.xodr"/>
41     <SceneGraph filepath="Databases/PEGASUS/PEGASUS_A01.opt.osgb"/>
42   </RoadNetwork>
43
44   <Entities>
45     <Object name="Ego">
51     <Object name="A1">
57   </Entities>
58
59   <Storyboard>
60     <Init>
61       <Actions>
62         <Private object="Ego">
79         <Private object="A1">
96       </Actions>
97     </Init>
98     <Story name="MyStory" owner="A1">
99       <Act name="MyAct">
100      <Sequence name="MySequence" numberOfExecutions="1">
175      <Conditions>
186      </Act>
187    </Story>
188    <End>
189  </End>
190 </Storyboard>
191
192 </OpenSCENARIO>

```

Abbildung 2: Ausschnitt einer OpenSCENARIO Datei von Andreas Biehn [8, (Downloads)]

OpenScenario ist aus mehreren Gründen gut geeignet um als Kommunikationsgrundlage zwischen Auto und Server zu fungieren. Zum einen handelt es sich dabei um eine Opensourcelösung, wodurch jeder Entwickler Weltweit selbstständig mit diesem Arbeiten kann. Zum anderen existiert wegen dem XML-basierten Dateiformat eine gute Lesbarkeit und Regelmäßigkeit Abfolge der Szenarien. Ein Szenario kann aufgrund

dieser Regelmäßigkeit einfach und schnell erstellt werden. Der letzte und größte Vorteil von OpenScenario ist es, dass man ein Szenario in dem Opensource Simulator "Carlaäbspielen kann.

Der im Folgenden vorgestellte Software-User-Pattern-Recognizer (SUPR) nutzt das OpenScenario-format zur Suche nach Software. Welche Suchvarianten es gibt und weitere Aufgaben des SUPR werden hier vorgestellt.

2.2.3 Software-User-Pattern-Recognizer [SUPR] im Rahmen der Bedarfserkennung

Das in diesem Abschnitt vorgestellte Konzept des Software-User-Pattern-Recognizer (kurz: SUPR) ist ein eigens ausgearbeiteter Baustein, welcher die Bedarfserkennung für intelligente Fahrzeuge beschleunigen und vereinfachen soll. Die Aufgaben des SUPR lassen sich in Aspekte der Bedarfserkennung sowie der Bereitstellung aufteilen, weshalb der SUPR in Kapitel 3.3 wieder aufgegriffen wird.

Eine Aufgabe des SUPR im Rahmen der Bedarfserkennung ist es, auf Anfrage hin eine Suche nach passenden Softwarepaketen durchzuführen (**1.**). Die zweite Aufgabe ist die regelmäßige Suche nach Software in der Umwelt/Region des Autos bzw. die Suche nach Softwarepaketen entlang zu Fahrender Strecken (**2.**). Das Maß der Rechenleistung auf Seiten des Servers ist indes höher als auf dem Fahrzeug. Eine mögliche Gegenüberstellung von Rechenleistung und Sendeleistung des Servers und des Autos kann zu behebende Defizite der Architektur aufdecken - die Optimierung des Systems ist jedoch nicht Teil des Forschungsseminars.

Der Server, auf welchem die Suche stattfindet, benötigt zwei Datenbanken: die eine enthält sämtliche Softwarepakete für intelligente Fahrzeuge, die andere eine große Menge an OpenScenario Dateien, welche zusätzlich Fremdschlüsselverweise auf ein oder mehrere Softwarepakete bereitstellen. Nur wenn diese Bedingung erfüllt ist, kann eine Suche durchgeführt werden.

1. Gezielte Suchanfragen

Gerät ein intelligentes Fahrzeug im Laufe seiner Fahrt in eine ihm unbekannte Situation, kann es diese mit Hilfe von OpenSCENARIO beschreiben. Das vom Auto erstellte Szenario wird an den Server geschickt, welcher dieses mit den auf der Datenbank liegenden Szenarien abgleicht. Ist die Suche abgeschlossen, wird eine Fallunterscheidung zwischen den folgenden Optionen getätigert:

A Es wird kein passendes Softwarepaket gefunden

Entweder existiert zu dem übergebenen Szenario keine Software oder möglicherweise wurde die im Szenario dargestellte Situation noch nicht zu einer Software gemapped. Dies sollte von speziell hierfür angestellten Arbeitnehmern überprüft werden, um so Dopplungen in der Softwareentwicklung zu vermeiden.

B Es werden (mehrere) passende Softwarepakete gefunden.

In diesem Fall gilt zu entschieden, welches Softwarepaket die besten Auswirkungen auf die Performance des Autos hat. Hierzu ist das Heranziehen diverser Kennzahlen ratsam um somit die Performance der unterschiedlichen Softwares untereinander zu vergleichen und eine fundierte Entscheidung treffen zu können. Eine beispielhafte Ausarbeitung dieser Entscheidungstreffung stellt Niklas Stelter in seiner Bachelorarbeit vor [29]. Am Ende einer Suche soll entweder eine einzelne oder keine Software vorgeschlagen werden.

Neben der vom Fahrzeug ausgehenden Suche nach einer bestimmten Software, kann dieses auch Vorschläge für Software von einem Server erhalten, welcher dauerhaft die Umwelt intelligenter Fahrzeuge analysiert.

2. Ortsbezogene Erkennung eines Softwarebedarfs Neben der Suche nach einer bestimmten Software soll der SUPR auch Suchen in der Heimatregion des Autos durchführen sowie auf zu Fahrenden Strecken. Diese Suchen basieren nicht auf OpenScenario-Dateien sondern auf Basis der Routenplanung[A] (Siehe Abbildung 1) und der Fahrtenhistorie des Fahrzeugs[B].

A: Suche auf Basis der Routenplanung

Wird vom Fahrer eine nicht oft oder noch gar nicht zurückgelegte Strecke vorgegeben, so soll der SUPR entlang der zu fahrenden Strecke häufig auftretenden Softwarebedarf identifizieren und dem Fahrer vor oder kurz nach Antritt der Reise diese Softwarevorschlägen. Daraus ergibt sich die Anforderung, zu jedem sich in der Datenbank befindlichen OpenScenario auch die geographische Position der Bedarfsentdeckung zu speichern. In Abbildung 3 ist folgende Situation dargestellt: Unser Auto hat als Ziel im Navigationssystem "Braunschweig" und befindet sich aktuell auf der A2. Die roten Boxen stellen den möglichen Bereich dar, in welchem der Server die Umweltanalyse betreibt. Wird eine Software entdeckt, überprüft das System die Relevanz für das eigene Fahrzeug und schlägt sie je nach dem vor oder nicht.

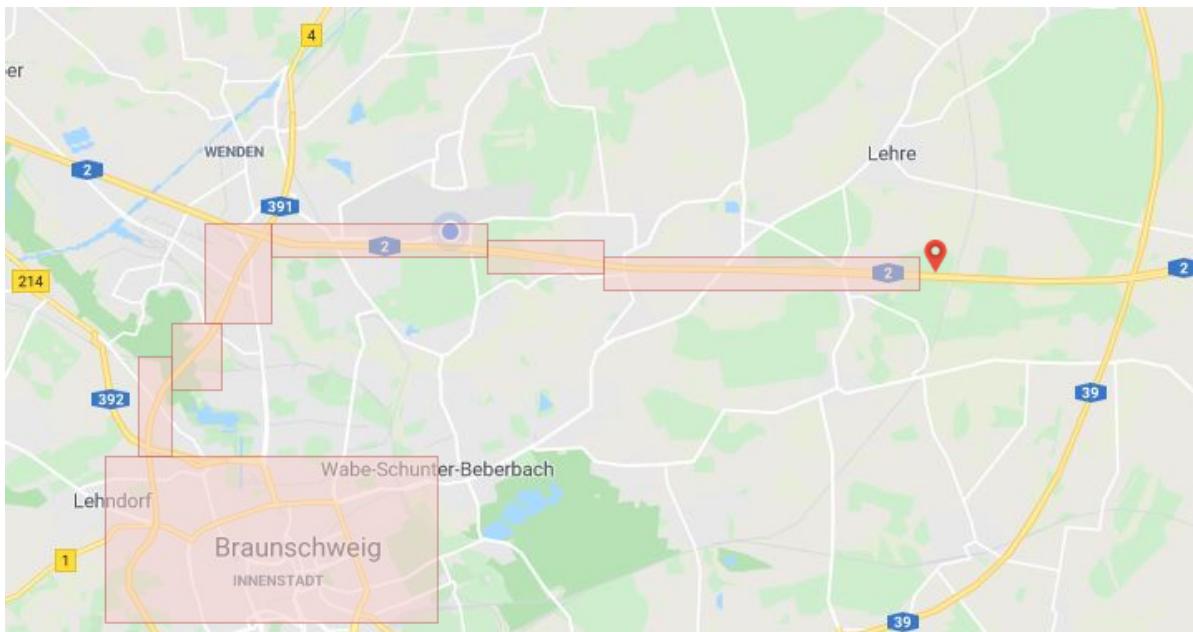


Abbildung 3: Umweltanalyse

B: Suche auf Basis der Fahrtenhistorie

Hierbei werden Muster in den zurückgelegten Strecken des Fahrers gesucht, um so dessen meist gefahrene Strecken zu identifizieren. Das Auto soll abschätzen können, wann der Fahrer welche Strecke zurücklegen wird. Vor dem jeweiligen Fahrtantritt soll die Suche nach Software identisch zu Variante A durchgeführt worden sein. Der Unterschied ist, dass die Suche hierbei selbstständig erfolgen soll, was ein besseres Nutzungserlebnis für den Fahrer schafft.

Ist die Suche (*egal ob gezielte Suche oder Umweltanalyse*) abgeschlossen, kann die Software dem Fah-

rer zum Kauf vorgeschlagen und bei Bestätigung für das Auto bereitgestellt werden. Das nächste Kapitel verdeutlicht den Umfang der Bereitstellung.

2.3 Bereitstellung von Software

Wurde der Bedarf einer neuen Software vom Fahrzeug festgestellt, muss diese im folgenden bereitgestellt werden. Zuvor muss der Softwarezulieferer die Sicherheit von Softwarepaketen verifizieren und eine sichere Kommunikation zwischen Servern und Fahrzeugen herstellen. Der Server muss alle vorhandenen Softwarepakete verwalten und auf Anfrage das am besten geeignete Softwarepaket für das jeweilige Fahrzeug identifizieren und ein Angebot an dieses schicken. Das Fahrzeug muss die eingegangenen Softwareangebote über die Mensch-Maschine-Schnittstelle zu einem Zeitpunkt anbieten, an dem die Kaufbereitschaft des Fahrers möglichst hoch ist. Hierzu bedarf es einem Softwaremanagement, welches die Angebote des Servers verarbeitet. Da die Grundvoraussetzung der Wertschöpfungskette eine sichere Kommunikation zwischen Auto und Softwarelieferanten ist, wird zunächst eine mögliche Architektur zur sicheren Kommunikation vorgestellt.

2.3.1 OTA-Aktualisierungen mit UPTANE

Der im folgenden vorgestellte Standard "UPTANE", stellt eine Architektur und Vorgehensweise für die sichere Kommunikation zwischen einem Server und einem Auto vor. Hierdurch können Software-Aktualisierungen an intelligente Fahrzeuge sicher verteilt werden [7]. Erste Schritte hierzu wurden 2010 getätig, als Justin Samuel, Nick Mathewson, Roger Dingledine und Justin Cappos "*The Update Framework*"(TUF) entwickelten. Dieses bildet den Grundbaustein für den späteren "UPTANE"Standard. Mittlerweile ist es Teil des Automotive Grade Linux Projekts und somit Teil der "Linux Foundation". Uptane stellt mittels einer Mehrschichtenarchitektur sicher, dass im Rahmen von Aktualisierungen keine Schädlingssoftware auf ein Auto gelangt. Zunächst wird die Architektur von Uptane dargestellt(Abb. 4).

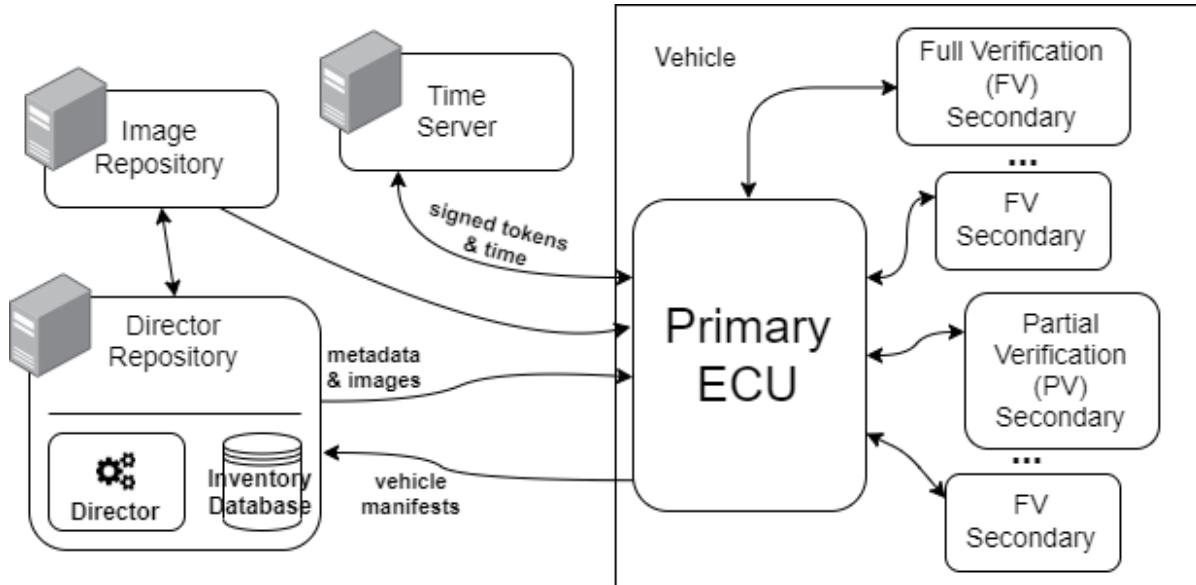


Abbildung 4: Architektur Design UPTANE [7]

Es ist vorab wichtig zu erwähnen, dass Uptane lediglich der Standard ist und keine offizielle Implementierung bereitstellt. Beispielhafte Implementierungen wären aktualizr¹, rust-tuf², Notary³ oder die OTA Community Edition⁴. Kommerziell entwickelte Systeme werden bereitgestellt von HERE Technologies⁵ sowie von Airbiquity⁶.

Die rechte Seite des Bildes stellt das Fahrzeug dar, die Elemente zur linken die Repositories. Diese Repositories sind Server und sie haben alle eine eigene wichtige Aufgabe. Der Time-Server ist dafür da, um ECUs über die aktuelle Zeit zu informieren, da viele ECUs keine Uhr haben [7]. Das Image Repository speichert jedes derzeit vom Lieferanten verteilte Image zusammen mit den Meta-Daten, welche zur Authentizität benötigt werden. Es nutzt Offline-Schlüssel um alle Metadaten zu unterschreiben" bzw. zu verifizieren, was einen hohen Sicherheitsvorteil darstellt. Das Director Repository entscheidet abhängig von den übergebenen Informationen des Autos genau, welche Images an die ECUs verteilt werden müssen. Ein Auto hat mehrere ECUs, welche sich in Speicherplatzgröße, Stromverbrauch und Aufgabenbereich unterscheiden. Die *Primary ECU* verwaltet und steuert die Installationen auf den anderen ECUs.

In dem ersten Schritt einer Aktualisierung schickt das Fahrzeug sein Manifest an das Director Repository. Dieses enthält Informationen darüber, welche Images aktuell auf dem Auto installiert sind. Das Director Repository entscheidet anhand dessen, welche Software auf dem Fahrzeug installiert/aktualisiert werden soll. Die Metadaten und die neuen Images werden zurück an die ECU geschickt. Hier findet zunächst eine Verifikation der neuen Images statt, bevor sie anschließend bei erfolgreichem Test installiert werden. Die Verifikation der ECUs kann entweder vollständig oder teilweise erfolgen. **Vollständig** heißt in diesem Kontext, dass die Größe der Software und die Hashes, welche die ECU über die Metadaten vom Director Repository erhält, identisch sind zu denen der Metadaten die vom Image Repository zur Verfügung gestellt werden. Bei der **teilweisen** Verifikation muss lediglich die Signatur der Metadaten des Director Repositories mit der Signatur der Metadaten vom Image Repository übereinstimmen.

2.3.2 Anpassung von UPTANE Architektur

Damit die durch UPTANE bereitgestellte Architektur die Erkennung und Bereitstellung neuer Fahrumfänge unterstützt, muss diese dementsprechend angepasst werden, damit neben dem Aktualisieren bereits installierter Software auch das Installieren neuer Software möglich ist. Hierzu wird die Architektur von UPTANE erweitert und angepasst.

Damit das Director Repository nicht nur bestimmen kann, welche Software eine Aktualisierung benötigt, sondern auch welche auf dem Fahrzeug installiert werden muss um den Bedarf zu decken, braucht es eine geeignete Kommunikationsgrundlage. Hierzu soll das zuvor vorgestellte OpenSCENARIO XML-Speicherformat verwendet werden. Um die Suche nach neuer Software für das Directory Repository zu ermöglichen, wird das in Abbildung 5 dargestellte *SScenario Repository*⁷ eingeführt. In diesem werden OpenSCENARIO-Dateien gespeichert, welche als Basis für die Suche nach Software dienen. Jede Datei hat hält mindestens **eine** Referenz

¹<https://github.com/advancedtelematic/aktualizr>

²<https://github.com/heartsucker/rust-tuf>

³<https://github.com/theupdateframework/notary>

⁴<https://github.com/advancedtelematic/ota-community-edition/>

⁵<https://www.here.com/products/automotive/ota-technology>

⁶<https://www.airbiquity.com/product-offerings/software-and-data-management>

in Form eines Schlüssels auf eine Software aus dem Image Repository.

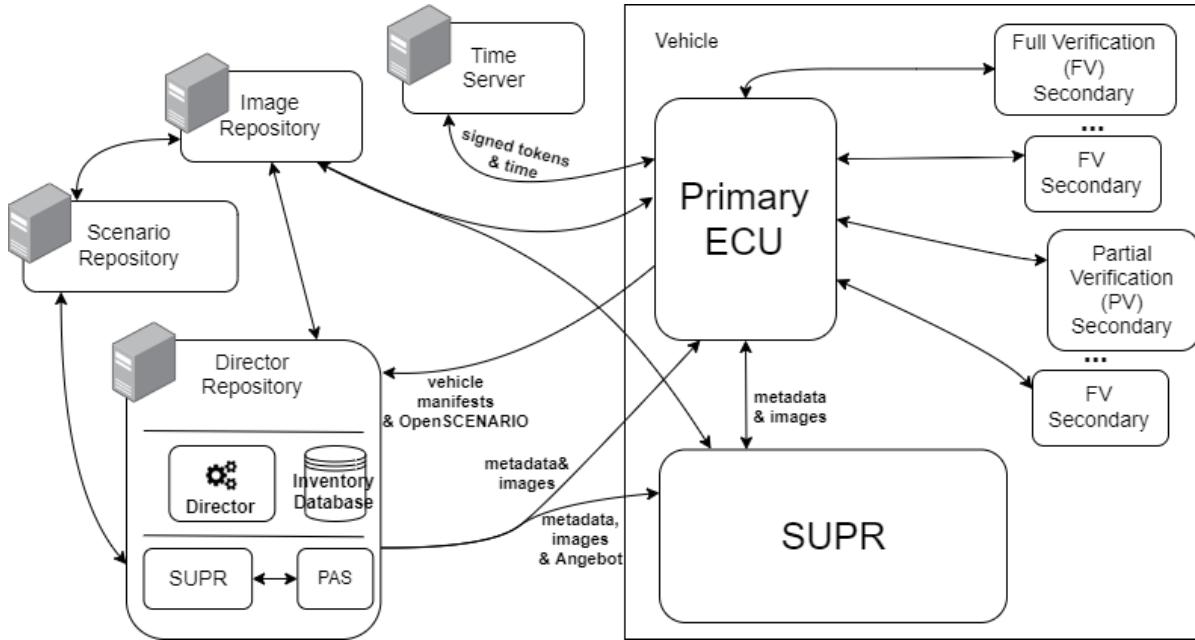


Abbildung 5: Angepasste Uptane Architektur

Dies bedeutet, dass die jeweilige Situation der OpenScenario-Datei von eben dieser Software bewältigbar ist. Erhält das Director Repository nun eine Anfrage für eine neue Software, wird diese anhand der mitgeschickten OpenSCENARIO-Dateien gesucht. Bei erfolgreicher Suche, soll diese in das Scenario Repository eingetragen werden. Die Suche führt der in Kapitel 2.3 eingeführte Software-User Pattern-Recognizer durch. In Kapitel 3.4 wird dieser um die Aspekte der Bereitstellung von Software erweitert.

Fahrzeughalter müssen neue Software kaufen, weshalb im Fall einer gefundenen Software zusätzlich zu den Metadaten und dem Softwarepaket (*Softwareimage*) noch eine Sammlung verkaufsbezogener Daten verschickt werden muss. Diese wird hier und im folgenden **Angebot** genannt. Ein Angebot muss alle möglichen Verkaufsarten (*Kauf, Miete, o.A.*) sowie dazu passende Preise beinhalten. Da die Preise und Verkaufsarten für Software je nach Automarke und -model variieren können sollen, wird ein Modul "pricing and sales", kurz PAS, zur Bestimmung des Preises und der Verkaufsart vorgeschlagen (*siehe Abbildung 5*). Das Director Repository ist nach wie vor dazu in der Lage, Softwareimages und Metadaten direkt an die Primäre Recheneinheit (ECU) zu schicken, wenn es sich nur um eine Aktualisierung bereits installierter Software handelt. Wird hingegen eine neue Software vorgeschlagen, so werden Image und Metadaten zusammen mit einem Angebot an den Software-User Pattern-Recognizer des Autos verschickt. Alternativ wird nur das Angebot verschickt um abzuwarten, ob der Kunde die Software tatsächlich haben will. Hierdurch würde das Image nicht unnötiger Weise verschickt werden, was unter anderem die Ressourcen schont. Der SUPR des Autos identifiziert unter stetiger Beobachtung des Fahrers, wann dieser einen Softwarevorschlag verarbeiten kann. Er stellt zum gegebenen Zeitpunkt Verbindung zu der Nutzeroberfläche des Wagens her, um so mit dem Fahrer zu kommunizieren und den Kaufvorgang abzuschließen (*Ob erfolgreich oder nicht ist hierbei egal*). Die Funktionsweisen und Aufgaben des SUPR im Auto unterscheiden sich von denen des SUPR im Director Repository. Aufgrund

dessen ist es wichtig eine Abgrenzung und Spezifizierung dieser vorzunehmen.

2.3.3 SUPR im Rahmen der Softwarebereitstellung

Das folgende komplettiert den in Kapitel 2.3 vorgestellten Software-User Pattern Recognizer (*SUPR*). Wie im ersten Teil wird auch im Rahmen der Bereitstellungskomponente des SUPR zwischen den Aufgaben auf Server und denen im Fahrzeug unterschieden.

SUPR im Director Repository

Die unterschiedlichen Suchvarianten welche der SUPR im Rahmen der Bedarfserkennung bereits auf dem Server ausführt, wurden in Kapitel 2.3 spezifiziert. Im Rahmen der Software Bereitstellung ist es die wesentliche Aufgabe, das Angebot und alle möglichen Preise festzulegen. Die verschiedenen Preise ergeben sich aus Kombination der einzelnen **Einflussfaktoren**.

Ein Angebot, welches vom Director Repository (siehe 3.2) an das SUPR des Fahrzeugs geschickt wird, spezifiziert die Preise für Software in Abhängigkeit der **Kaufart**, der **voraussichtlichen Laufzeit** des Mietverhältnisses (bei Miete), der Menge der Software welche der Fahrzeughalter in dem Moment akquirieren möchte und den eigentlichen Herstellungskosten der Software. Die folgende Tabelle definiert die Werte, welche die einzelnen Einflussfaktoren annehmen können.

Wert	Beschreibung
Kaufart	
Kauf	Der Fahrer akquiriert die Software dauerhaft. Eine Reklamation ist möglich, wenn diese ihre Aufgabe nicht korrekt ausführen würde.
Leihe	Der Fahrer legt einen Zeitraum(<i>von .. bis</i>) fest, für welchen er diese Software akquirieren möchte. Für diesen bezahlt er im Voraus und die SW wird bei Ablauf des Zeitraums deinstalliert.
Miete/Abo	Der Fahrer wählt die Dauer einer Mietphase, an welche sich der Preis anpasst. Das Mietverhältnis wird nach Ablauf dieses Zeitraums aufrecht erhalten. Das heißt es entstehen laufend Kosten bis es vom Fahrer beendet wird.

Zeitraum/Dauer	
Permanent	Der Zeitraum ist nur dauerhaft, wenn die Software gekauft wird. Hierbei tritt der höchste Preis auf.
Lang	Sechs Monate oder mehr. Auf einen Tag heruntergerechnet der günstigste Miet- oder Leihpreis.
Mittel	Sechs Wochen bis zu sechs Monate. Auf einen Tag heruntergerechnet ein wenig teurer als der des Langen.
Kurz	Acht Tage bis sechs Wochen. Auf einen Tag heruntergerechnet ein wenig teurer als der des Mittleren.
Einmalig	Ein bis sieben Tage. Auf einen Tag heruntergerechnet das teuerste Angebot.

Menge	
Einzeln	Es wird nur eine Software zu dem Zeitpunkt angeboten. Hat keinen Einfluss auf den Preis.
Mehrere	Software wird im Bundle gekauft, was sich positiv auf den Preis auswirkt. Ein Bundle muss vom SUPR im Fahrzeug erstellt werden.
Flottenkauf	Es wird Software für mehrere Fahrzeuge gekauft. Dies kann für Unternehmen Sinnvoll sein, die Dienstwagen anbieten oder auch Autovermietungen etc.

Weitere Einflüsse auf den Preis	
Herstellungskosten	Die Kosten die im Laufe der (Weiter-)Entwicklung und der Wartung dieser Software entstanden sind.
Beliebtheit	Ist eine Software beliebt bzw. gefragt, steigt der Preis dieser. Die Beliebtheit kann entweder global gemessen werden oder regionsabhängig. Der Preisanstieg sollte dabei nicht zu hoch sein.
Mächtigkeit	Je mehr eine Software "kann", desto teurer sollte diese sein.

Nachdem der SUPR des Director Repositorys eine den Bedarf deckende Software gefunden hat, muss er die Referenz auf die Datei an das PAS-Modul (*siehe Abbildung*) schicken. Das PAS-Modul soll anhand der Software, einiger Informationen zum Kaufverhalten der Fahrer des Fahrzeugs sowie dem generellen Bedarf der Software ein personalisiertes *Angebot* erstellen und anschließend an das Fahrzeug schicken.

SUPR im Fahrzeug

Neben dem Server-seitigen SUPR hat auch der SUPR des Autos wichtige Funktionen im Rahmen der Bereitstellung von Software. Ist das Angebot im Auto angekommen, identifiziert dieser einen geeigneten Verkaufszeitpunkt und stellt zu diesem das personalisierte Angebot über die Mensch-Maschine Schnittstelle (MMS) dar. Dies meint den Zeitpunkt, an dem der Verkauf oder die Leih von Software möglichst wahrscheinlich ist. Hierzu ist eine Beobachtung des Fahrers nötig, um anhand von dessen **Eigenschaften** wie bspw. Mimik, Gestik, Sprache oder dem Fahrverhalten das Stress-, Freude, oder Angstlevel zu deuten, aber auch um Müdigkeit oder Ablenkung feststellen zu können. Vor allem Faktoren wie Stress, Freude und Angst beeinflussen unsere Meinung und Entscheidung. "Vgl. [27, S.44]. So wird eine Software eher nicht gekauft, wenn der Fahrer in eine stressige Verkehrssituation überblicken muss, zum Beispiel wenn es dicht benetzt ist und man zur Zeit in einer unbekannten Gegend Auto fährt. Hingegen beeinflusst Freude einen eher dazu, einen Kauf zu tätigen. Angst ist für den Anwendungsfall des Verteilens neuer Fahrfunktionen divers zu deuten. Wenn der Fahrer Angst vor der Strecke hat die vor ihm liegt, muss der SUPR dies anders bewerten als Angst die Aufgrund anderer Einflüsse entsteht. Diese Beobachtung entspricht der dritten Richtlinie Mensch-zentrierter autonomer Fahrzeuge nach Lex Fridman. [20, S. 3] Nach der fünften Richtlinie Fridmans, soll ein Auto von dem Moment dem ersten Einstiegs des Fahrers auf diesen personalisiert werden. [20, S. 5] So ist es Sinnvoll, würde der SUPR persönliche Eigenschaften in seine Entscheidungsfindung zum geeigneten Verkaufszeitpunkt einfließen lassen.

Ein System eines Fahrzeugs soll laut Fridman Daten-orientiert sein [20, S. 3]. Das heißt, dass ein Fahrzeug aus den aufgenommenen Daten lernen soll. Fusioniert man die bei der Fahrerbeobachtung aufgenommene Daten mit Daten über den Verkaufserfolg, kann der SUPR aus seinen Handlungen lernen und sich so optimieren.

Hierdurch werden mögliche den Kauf negativ beeinflussende Faktoren identifiziert (zB. Stress) und der Fahrer so besser kennengelernt. Die Kaufbereitschaft des Fahrers soll in Folge dessen möglichst oft korrekt eingeschätzt werden.

Ist der Zeitpunkt bzw. Zeitrahmen identifiziert, stellt der SUPR über die Mensch-Maschinen-Schnittstelle das personalisierte Angebot dar. Der dargestellte Inhalt wird maßgeblich beeinflusst von den Aspekten der Verkaufspräzesspsychologie. Um die wesentliche Aufgabe des SUPR (*Beobachtung & Kommunikation mit Fahrer, Vermarkten von Software*) sinnvoll zu modellieren, werden die Einflussfaktoren für den dargestellten Inhalt der MMS anhand der Prinzipien der Verkaufspolitik nach Markus Reinke [25] erläutert.

Um einen Kunden vom Erwerb eines Produkts zu überzeugen, muss diesem etwas angeboten werden wodurch ein offenes Bedürfnis befriedigt. Dazu muss herausgefunden werden, welches Produkt geeignet ist, damit es so anschließend angeboten werden kann. Wie der SUPR die Bedürfnisse von Fahrern identifiziert ist in Kapitel 2.3 nachzulesen. Was bei dem Anbieten einer Software beachtet werden muss, wird in diesem Kapitel abgehandelt.

Kunden sind unterteilbar in Plus-, Chancen- und Minus-Kunden. [25, S. 10ff.] Minus-Kunden werden ein Produkt von Beginn an nicht kaufen wollen, so gut der Erwerb auch gestaltet sein wird. Plus-Kunden werden ein Produkt voraussichtlich kaufen. Um die Chancen-Kunden von einem Erwerb zu überzeugen, werden im Absatz von Unternehmen diverse Prinzipien angewendet. Es kann also die Schlussfolgerung gezogen werden, dass so gut der SUPR seinen Fahrer auch kennt und wie gut Produkte den Bedarf auch abdecken, der Kunde dennoch nicht immer eine Software erwerben wird. Um die Anzahl an Käufern möglichst hoch zu halten, soll der persönliche Nutzen den die SW für den Kunden hat dargestellt werden. So kann die Situation simuliert und dargestellt werden, in welcher das Fahrzeug den Softwarebedarf festgestellt hat.

Das Differenzprinzip [25, S. 19fff.]

Nach dem Differenzprinzip sollen unterbewusste Reize gesetzt werden, die zum Erwerb leiten. Zum Beispiel wird mit der teuersten Variante des Produkts geworben, damit der Kunde bei näherem betrachten des Angebots die tieferen Preise, also ein "kleineres Übel", entdeckt und sich darüber erfreut. Dabei soll nicht die Frage **ob** der Kunde bei einem kaufen möchte, sondern **was** der Kunden von einem kaufen will.

Stellt der SUPR das Angebot dar, soll eine Mietfunktion im Fokus stehen, zudem deutlich gemacht werden. Die Option Software Kaufen ist kleiner darzustellen zusammen mit dem jeweiligen Preis. Um dem Kunden die Möglichkeit zu lassen die Software nicht zu erwerben, muss ein dezenter Knopf in der Nutzeroberfläche sein, welcher den Angebotsprozess beendet. Durch die Unauffälligkeit kann der Kunde zum Kauf gelenkt werden bzw. zu der Frage **was** man kaufen möchte, nicht **ob**. Um einen unterbewussten Reiz zu setzen, kann eine Karte dargestellt werden wie häufig die Software in der Umgebung installiert wurde. Dies sollte allerdings nur geschehen, wenn die Nachfrage tatsächlich auffällig hoch ist. Hiermit würde auch der Nachahmungseffekt abgedeckt werden (*weiter unten*).

Das Do-ut-des-Prinzip [25, S. 33fff.]

Nach diesem Prinzip, wird "gegeben, damit du gibst". Der Kunde wird durch beispielsweise Gratis-Proben angelockt um zum Erwerb bewegt zu werden. Will er dennoch nichts kaufen, kann um Weiterempfehlung gebeten werden. Dieser Bitte wird der Kunde vrs. nachkommen, da er im Vorfeld etwas "gratist erhalten hat.

Die *SZwei Schritte vor, einer Zurück Taktik*" hierbei anzuwenden ist sinnvoll. Das heißt, dass dem Kunden zuerst die teuersten Produkte gezeigt werden, um ihn anschließend auch hier mit niedrigen Preisen anderer Produkte beeindrucken und halten zu können. Es sollte hierbei auch nicht das Ziel das teuerste Produkt zu verteilen, sondern das eigentliche Ziel ist eines mit tieferem Preis.

Der SUPR kann dieses Prinzip anwenden, in dem er dem Kunden zu neu erworbener Software eine Test-Version einer anderen Software schenkt", welche nach einem bestimmten Zeitraum (4 Wochen) abläuft. Diese Software sollte einen bestehenden Bedarf der Kunden abdecken.

Das Konsequenzprinzip

Konsequenz im Handeln wird in den meisten Kulturen als eine positive Charaktereigenschaft gewertet, strahlt Sicherheit aus. Zudem schafft es Vertrauen bei Kunden. Konsequenz kann unter anderem mittels wenn-dann-Fragen ausgestrahlt werden, also "Wenn wir Ihnen XY bieten, kaufen Sie dann?" Der Kunde würde bei Erfüllung seines Bedarfs eher kaufen. Und da der von ihm benannte Bedarf abgedeckt ist, wird er so von einem frühen Commitment überzeugt. Es sollen die Wünsche und Prioritäten der Kunden erkannt werden und anschließend das Produkt mit Fokus auf diese beworben werden.

Dieses Prinzip sollte vor allem verwendet werden, um einen "Fuß in die Tür zu bekommen" und nicht um das teure Produkt zu verkaufen. Sinnvoll ist es, nach dem Motto "Kleinvieh macht auch Mistfuß verkaufen. Der SUPR tut dies zum einen indem er die Situation, in welcher der Bedarf festgestellt wurde, auf der MMS darstellt und den Kunden so erinnert. Wenn der Kunde noch gar keine oder nur wenig Software erworben hat, soll ihm immer die Möglichkeit einer kostenlose Probe-Woche geboten werden. Dies soll bis zu fünfmal geschehen, danach wird ohne weiteres keine Gratis Software mehr angeboten.

Das Nutzen des Nachahmungseffekts [25, S. 67fff.]

"Der Mensch ist grundsätzlich ein Gemeinschaftswesen und achtet daher sehr darauf, was andere von ihm denken, sowohl in der breiten Öffentlichkeit als auch in seinem engen persönlichen Umfeld. Markus Reinke (Vgl.) [25, S. 67]

Dieses Verhalten, der sogenannte **Nachahmungseffekt**, kann durch das Einsetzen verschiedener Techniken provoziert werden. Bei der **Zeugenumlastung** lässt sich der Kunde bestätigen, dass ihr Unternehmen und Produkt "gut ist" indem er von anderen positives darüber in Erfahrung bringt. Eine Variante, wie diese vom SUPR angewendet werden kann, wurde zuvor bereits dargestellt. Bei der **Referenztechnik** werden die Stammkunden eines Unternehmens gebeten Empfehlungsschreiben zu erstellen mit welchen im folgenden geworben werden kann. Der SUPR kann hierzu ein Bewertungssystem einführen und Nutzer können die Bewertung anderer Fahrer lesen und sich so eine Meinung der Software einholen. Dabei ist es sinnvoll, einige vorgefertigte Antwort/Bewertungsmöglichkeiten selber zur Verfügung zu stellen. Eine dritte Technik ist das **Empfehlungsmarketing**, bei welchem Kunden gebeten werden uns an Freunde und Verwandte weiter zu empfehlen. In den Zeiten von Social Media bietet es sich an, eine "TeilenFunktion zur Verfügung zu stellen. Der Fahrer soll Teilen können, wie viele Kilometer er in welcher Zeit zurückgelegt hat, wie viele davon autonom bewältigt wurden und welche Softwarepakete genutzt wurden.

Um eine hohe Resonanz herzustellen, ist es sinnvoll dem Kunden ähnlich zu sein, sympathisch zu wirken, Lob & Anerkennung zu verteilen als auch andere zu unterstützen. Da es ungewohnt ist, Lob oder anderes von

einem Computer zu erhalten, der SUPR aber dennoch eine hohe Resonanz erzielen soll, sollte den Kunden eine Bezugsstelle gegeben werden. Eine Referenz aus der Wirtschaft hierzu stellt "Meet Olli" dar [2]. Olli ist das Maskottchen des UserInterfaces (UI) von "Local Motors" (LM). LM stellt einen autonom fahrenden Bus her, mit welchem die Insassen über eine UI interagieren können. Olli hat "nur" zwei Augen und einen Mund, stellt hierdurch aber einen vermenschenlichen Bezugspunkt für den Fahrer und die Insassen dar und gewinnt so das Vertrauen der Insassen.

Darüber hinaus soll der Kunde nicht von den Angeboten "erschlagen" werden, sondern der SUPR muss diese dosiert vorlegen. Wenn der Kunde kein Interesse hat, ist dies zu respektieren und er soll damit nicht weiter konfrontiert werden. Der SUPR soll nur die tatsächlich benötigte Software bewerben - ein einmalig aufgetretener Bedarf fällt da nicht drunter (*Die Ausnahme ist, der Kunde ist ein absoluter Plus-Kunde*). Es muss genügend Werbung betrieben werden damit möglichst viel Software verkauft wird. "Klassische" Werbung auf der MMS ist nicht Sinnvoll, sondern eher Werbung in Form von Gratis-Proben von Software-Paketen. Nutzt der Fahrer eine dieser Proben und die SW kommt zum Einsatz, kann der SUPR den Fahrer über die MMS darüber in Kenntnis setzen, wodurch er vor Augen geführt bekommt, dass er diese Software braucht.

Ein Auto wird allerdings nicht nur von einer Person gefahren, sondern möglicherweise von mehreren Familienmitgliedern, Mitarbeitern oder anderen Mietern im Falle von Carsharing. Das SUPR muss dazu in der Lage sein, diese einzelnen Personen auseinanderhalten zu können um keine falschen Entscheidungen zu treffen.

Wird dem SUPR ein Softwarevorschlag mit Angebot zugeschickt, soll dieser bis das Angebot angezeigt werden kann, diese Software schon anfangen herunterzuladen. Dadurch kann die Software, wenn sie erworben wird, bereits während der Fahrt installiert werden, was den Verkauf schneller und Nutzerfreundlicher gestaltet.

2.3.4 Die Mensch-Maschine-Schnittstelle

Die Mensch-Maschine-Schnittstelle ist das System, über welches der Fahrer und das Auto mit einander interagieren. Im Kontext intelligenter Fahrzeuge handelt es sich hierbei meist um einen Bildschirm, mitunter haben Fahrzeuge auch einen integrierten Sprachassistenten.

Das Bereitstellen neuer Fahrfunktionen schließt mit ein, dass ein Fahrzeug zum Zeitpunkt A selbstständig Fahren kann und zum Zeitpunkt B nicht. Zwischen diesen Zeitpunkten muss eine Übergabe der Fahraufgabe von dem Fahrzeug an den Fahrer erfolgen. Dabei soll der Fahrer des Fahrzeugs über die momentane Verkehrslage ausreichend in Kenntnis gesetzt werden, um während und nach der Übergabe der Fahraufgabe die Sicherheit zu wahren. Der Begriff ausreichend definiert sich dabei wie folgt:

Die Inkenntnissetzung gilt als vollständig, wenn die Mensch-Maschine-Schnittstelle alle Prinzipien von S. Debernard et Al. [26] erfüllt.

Das zu entwickelnde Display soll Nutzer-zentriert (User-Centered) entwickelt werden, um die Anforderungen möglichst vieler Nutzer erfüllen können und zusätzlich den Faktor der "Vermenschlichung" erhöhen. Die geschaffene transparente Straßenführung über das Display eines Fahrzeugs, soll zusätzlich durch Audio-Assistenten unterstützt werden. Die MMS wird so durch ein audiovisuelles System.

Die Schnittstelle soll sich sowohl visuell als auch auditiv an die im Fahrzeug sitzenden Personen anpassen. So

kann zum Beispiel die Schriftgröße größer sein, wenn eine Person im Rentenalter das Fahrzeug betritt oder ähnliches. Mögliche Features der Personalisierung (*z.B. größere Icons, dunkles Design*) werden mit Hilfe der Personas 2.4.1 in der Bachelorarbeit erarbeitet. Im generellen soll der Nutzer auf dem Display eine Simulation des eigenen Autos sehen zusammen mit zusätzlicher Information gemäß den Prinzipien nach Debernard et. Al. Das Design des User-Interfaces kann vom Fahrer angepasst werden, wobei die intuitive Bedienung sich nicht verschlechtern soll. Durch die Vermenschlichung des MMS, einer Transparenten Straßenführung, der Möglichkeit die Steuerung jederzeit zu übernehmen sowie der höflichen Kommunikation mit dem Fahrer, wird das Vertrauen des Fahrers zu dem Fahrzeug gefördert [2], was den Kauf von Software wahrscheinlicher macht.

2.4 Technologie- und Methodik-Scouting

Um auf Grundlage der dargestellten Rahmungen für die Bedarfserkennung und Bereitstellung neuer Fahrumfänge für intelligente Fahrzeuge einen Prototypen zu entwickeln, wird abschließend ein Technologie- und ein Methodik-Scouting durchgeführt werden. Das Kapitel der Arbeitsmethodik spezifiziert Den Zyklus des User-Centered-Designs und stellt dar, weshalb dieser für die Entwicklung des Prototypen die richtige ist. Das anschließende Technologie-Scouting stellt Entwicklungsumgebungen, Frameworks und Programmiersprachen vor, welche in der Entwicklung genutzt werden.

2.4.1 Arbeitsmethodik: User-Centered-Design

Um die Nutzeroberfläche für die Zielgruppen ansprechend zu gestalten, wird bei der Erarbeitung dieser nach den Prinzipien des User-Centered Designs (*UCD*) gehandelt. Ins deutsche übersetzt bedeutet dies "Benutzerzentriertes Design bzw. Benutzerorientierte Gestaltung eines Produkts. Es beschreibt einen Designprozess bzw. ein Entwicklungsverfahren, auf welches der Endnutzer (Fahrer) schon von Anfang an Einfluss nimmt" (Vgl. [15, S.763]). Der User wird in die Phasen der Entstehung einer Benutzungsschnittstelle integriert was zur Folge hat, "dass der Aufbau, die Inhalte und deren Form sowie das Design des Endproduktes maßgeblich von den Bedürfnissen, Erwartungen und dem Verständnis der User bestimmt wird" (Vgl. [1]). Des weiteren kann durch das Einbeziehen von Kunden die Qualität optimiert werden (Vgl. [12, S. 14]).

"Typisch für einen UCD-Prozess zum Entwerfen von Webanwendungen mit optimaler User Experience sind [jedoch] die Prozessschritte Analyse, Konzeption, Umsetzung/ Design, Evaluierung und Optimierung." (Vgl. [11])

Im Rahmen der Analyse werden die Anforderungen an das System erstellt und zusammengeführt. Hierzu können Personas erstellt werden (siehe: 2.4.1) und aus eben diesen können Anforderungen erstellt werden. Die Analyse soll zudem "Usability" [14] Ziele festlegen, anhand welcher in der "Evaluierung und Optimierung" die Güte der Nutzeroberfläche gemessen werden kann [11]. Während der Konzeption soll das Verständnis der Benutzer und deren Bedürfnisse hinsichtlich der User Experience auf die Benutzeroberfläche übertragen werden [11, (ebd.)]. Die Konzeption endet mit dem erstellen eines Prototypen. Dieser wird in der Dritten Phase (Design) durch ein konsistentes, ansprechendes und klares Grafik Design" (Vgl. [11]) erweitert, was Probleme löst und eine Intuitive Bedienung unterstützt. Die Phase der "Evaluierung und Optimierung" wird das Produkt auf Probleme wie Sicherheit oder eine schlechte Bedienbarkeit hin untersucht. Hierdurch werden

Mängel entdeckt welche in den nächsten Zyklus der Entwicklung einfließen.

Wie hoch der Grad der Usability und der User-Experience ist, lässt sich wie zuvor erwähnt anhand von Personas ableiten. Im Folgenden wird daher erklärt, was Personas sind und es werden eigene Personas für das Projekt erstellt.

Personas

Bei Personas handelt es sich nun um fiktive Personen, also hypothetische User, mit individuellen Eigenschaften [9,], welche ebenso eine reale Person haben könnte. Je ähnlicher die Personas also der Zielgruppen des Unternehmens/des Produktes sind, desto effektiver ist die Verwendung dieser.

Personas sind nicht nur als potentielle Zielgruppe zu sehen - sie haben zudem die Aufgabe, dass sich das Entwicklerteam in die Position des Nutzers versetzen kann. Dazu werden Personas zunächst nach Kategorien wie "Geschlecht/Alter" aufgeteilt und anschließend wird jedes Profil mit Merkmalen und Eigenschaften gefüllt. Welche Informationen unter anderem zu Personas hinzugefügt werden, zeigt die folgende Tabelle.

Vor- und Nachname	Geburtsdatum/Alter
Foto der Person/Aussehen	Herkunft/Wohnort
Sprachkenntnisse	Beruf; Berufserfahrung in Jahren
Bildung/Ausbildung	Familienstand
Interessen und Hobbys	Fähigkeiten/Behinderungen
Sicherheitsrisikofaktoren für den Straßenverkehr	Abneigungen
Erfahrungen mit Technik (Meidenkompetenz)	Vorlieben
Fahrerfahrung	Kaufbereitschaft
Auto; Wenn ja: Besitzverhältnis?	

Anhand dieser möglichen Informationen werden im folgenden die Personas für dieses Projekt erstellt.

Name / Soziographische Daten	Hobbys & Interessen	Anwendungsfallbezogenes
 <ul style="list-style-type: none"> • Dietmar Müller, 68 Jahre (Bild: [10]) • Loppersum, Ostfriesland • Deutsch • Gelernter Maurer • Verheiratet, 3 Erwachsene Kinder 	<ul style="list-style-type: none"> • Rentner; Taxiunternehmer (seit 20 Jahren aktiv) • Fischen, Wandern • Handwerklich begabt, Hobbygärtner • Diabetiker • idR. Minus-Kunde • Geizig 	<ul style="list-style-type: none"> • Führerschein seit 47 Jahren • Vertraut Technik nicht • Schlechte Sehkraft • Ängstlicher Autofahrer • Besitzt eigenen Neuwagen
 <ul style="list-style-type: none"> • Jake Schneiders, 52 Jahre (Bild: [3]) • Washington, USA • Englisch, Russisch (Fließend) • Ausgebildeter Cyberhacker • Geschieden, ein Kind (geteiltes Sorgerecht) 	<ul style="list-style-type: none"> • Leutnant beim Militär • Jagen, Baseball Trainer • Kindersorgerecht unter der Woche • Workaholic • Chancen-Kunde 	<ul style="list-style-type: none"> • Führerschein seit 27 Jahren • Vertraut sehr auf Technik • guter, sicherer Autofahrer • Setzt Wert auf gute Bedienbarkeit • Hat gerne die Kontrolle über Systeme • Fährt einen Wagen des Militärs • Arbeitet viel am Laptop

 <ul style="list-style-type: none"> • Chi Nguyen, 24 Jahre (Bild: [13]) • Rom, Italien • Vietnamesisch, Italienisch(Fließend), Englisch (Fließend) • Studentin (Geschichte) • Single 	<ul style="list-style-type: none"> • Studentin, Stadtführerin in Rom • Joggen, Roadtrips • Influencer • Plus-Kundin 	<ul style="list-style-type: none"> • Führerschein seit 2 Jahren • Vertraut sehr auf Technik • unsichere Autofahrerin • Ist mit viel Technik aufgewachsen • Mag es, ihre Apps zu personalisieren • Nimmt Teil an Car-Sharing
--	---	---

Die erstellten Personas werden bei der Erstellung des Prototypen herangezogen um Anforderungen an das System. Jede einzelne ermöglicht es, bestimmte Schwierigkeiten und Herausforderungen genauer darzustellen. So soll Dietmar Müller verdeutlichen, dass ein Kunde möglicherweise doch vom Kauf einer SW überzeugt werden kann, obwohl er idR. ein Minus-Kunde ist. Durch ihn kann auch dargestellt werden, welche weiteren Einflussfaktoren es neben Stressö.Ä. gibt, wie bei seiner Diabetes Erkrankung.

Jake Schneiders soll die Art Person darstellen, welche neugierig ist, die Sicherheit eines System testen möchte und hierzu auch selber in der Lage ist. Durch seine technische Affinität kann er Systeme leicht Verstehen und neue Anforderungen erstellen, die voraussichtlich leicht zu implementieren sind.

Chi Nguyen ist eine sehr feminine Studentin welche bereits vielerorts gewohnt hat. Durch ihr Dasein als Influencerin hat sie ein großes Netzwerk an Followern, mit welchen sie ihre Erfahrungen die sie während der Autofahrt sammelt Teilen kann.

Da alle Personas in unterschiedlichen Ländern an dem Straßenverkehr teilnehmen, sind die Einflussfaktoren zur Erstellung von Anforderungen sehr vielfältig, was sich positiv auf die Entwicklung auswirken kann.

2.4.2 Tech-Scouting

Damit die Entwicklung des Prototypen möglichst simpel sein wird, wird im Folgenden ein Technologie-Scouting durchgeführt. Es soll die Vorteile aufzeigen, welche ausgewählte Entwicklungsumgebungen, Programmiersprachen und Frameworks für die Entwicklung des Prototypen haben.

Im Rahmen eines Prototypen für die Erkennung und Bereitstellung neuer Fahrfunktionen für intelligente Fahrzeuge ist es nötig, ein auf der Straße fahrendes Fahrzeug zu Simulieren und dazu eine passende Nutzeroberfläche zu haben, welche als Mensch-Maschine Schnittstelle des Fahrzeugs agiert. Für das Simulieren einer Straßenverkehrssituation wird der OpenSource Simulator **Carla**⁷ verwendet. Mit Carla ist es möglich, das eigene Auto zusammen mit anderen Fahrzeugen, Radfahrern, Fußgängern, Hindernissen uvm. auf der Straße darzustellen. Autos können mit Sensoren ausgestattet werden und die aufgenommenen Daten können mittels der Python-API ausgelesen werden. Der Ablauf einer Simulation wird in einem Python-Script spezifiziert. Eine alternative Möglichkeit zur Erstellung einer Simulation ist das einbinden einer OpenSCENARIO-Datei. Es existiert ein Aufsatz auf Carla, wodurch der Simulator den in einer solchen Datei dargestellten Ablauf selber darstellen kann.

Für die Erstellung von Python Skripts wird "**PyCharm**"⁸ als Entwicklungsumgebung gewählt. PyCharm ermöglicht eine intuitive und einfache Anbindung an GIT. Da die aus Carla ausgelesenen Daten verarbeitet werden müssen um zu erkennen wann das Auto nicht mehr selbstständig fahren kann, ist die Entwicklung eines Servers notwendig. Neben der Analyse der Simulation soll dieser als Kommunikationszentrale zwischen eben diesem und der Mensch-Maschine-Schnittstelle dienen. Zur Entwicklung des Servers soll das ebenfalls von JetBrains entwickelte **IntelliJ IDEA**⁹ dienen. Auch IntelliJ bietet eine einfache Anbindung an GIT an - zusätzlich ist die Einbindbarkeit von **Maven**¹⁰ zum verwalten von *Dependencies* ein leichtes.

Für die Entwicklung der Mensch-Maschine-Schnittstelle wird eine Android App entwickelt - für diese wird **Android Studio**¹¹ verwendet. Android Studio ermöglicht neben der visuellen auch eine textuelle Bearbeitung von Nutzeroberflächen. Es ist die von Google empfohlene Entwicklungsumgebung und ermöglicht das erstellen von Apps massiv.

Neben Python wird zum entwickeln von Server und MMS hauptsächlich die Programmiersprache Java (Version 1.10) verwendet. Java wurde an der Universität Oldenburg gelehrt und es wird unter anderem zum entwickeln von Android Apps verwendet.

2.5 Ausblick und erstes Konzept der praktischen Umsetzung

Nachdem in den vorherigen Kapiteln einzelne Aspekte der Wertschöpfungskette erläutert wurden, soll es abschließend einen Ausblick auf die Bachelorarbeit geben und ein erstes technisches Konzept des Prototypen vorgestellt werden. Im Laufe der Arbeit haben sich einige zu erläuternde Aspekte aufgetan, welche aus Platzgründen erst in der Bachelorarbeit behandelt werden.

Im Rahmen dieser soll zunächst ein erster Zyklus des UCD durchgeführt werden, um so die Nutzeroberfläche der Mensch-Maschinen-Schnittstelle zu erarbeiten. Dabei sollen zusätzlich mögliche Personalisierungsfeature benannt werden. Weiterhin sollen die anderen im Folgenden vorgestellten Teilsysteme entwickelt werden. Unter anderem soll ein Suchalgorithmus für den SUPR des Servers erarbeitet werden sowie eine Gegenüber-

⁷<http://carla.org/>

⁸<https://www.jetbrains.com/de-de/pycharm/>

⁹<https://www.jetbrains.com/de-de/idea/>

¹⁰<https://maven.apache.org/>

¹¹<https://developer.android.com/studio>

stellung von Rechenleistung und Sendeleistung des Autos bzw. des Servers erfolgen. Es werden die einzelnen Bausteine einer Wertschöpfungskette (zur Erkennung und Bereitstellung neuer Fahrumfänge intelligenter Fahrzeug) benannt, erläutert und im Prototypen veranschaulicht. Im Folgenden wird für diesen ein erstes technisches Konzept vorgestellt.

Erstes technisches Konzept des Prototypen

Abbildung 6 zeigt eine erste Architektur des Prototypen. Dieser ist unterteilt in das **Fahrzeug**(unten) und den **Server**(oben). Das Fahrzeug ist in drei Teilsysteme aufgeteilt. Der "Carla Python Skript Client" ist das Modul in dem die Simulation stattfindet. Entscheidungen bzgl. der Fahraufgabe werden in diesem getroffen. Die **Mensch-Maschine-Schnittstelle** ist eine Android-basierte App. Sie wird über einen Emulator bedienbar sein und soll unter anderem Informationen aus dem Carla Simulator darstellen. Damit dies möglich ist, wird der dritte Baustein des Fahrzeugs eingeführt: der SUPR.

Der SUPR des Fahrzeugs soll die im Forschungsseminar definierten Aufgaben durchführen können und dient zusätzlich als Kommunikationseinheit. Er verbindet die Carla Simulation und die Mensch-Maschine-Schnittstelle, sodass eine interne Kommunikation im Fahrzeug möglich ist. Des Weiteren dient der SUPR als Kommunikationseinheit zum Server. Er ist für die Installation eingehender Softwarevorschläge zuständig und muss entsprechende Updates an den Python Client senden. In der Uptane Architektur waren hier die ECU's eines Autos angedacht. Da diese in dem ersten Konzept nicht auftauchen, steuert dies der SUPR.

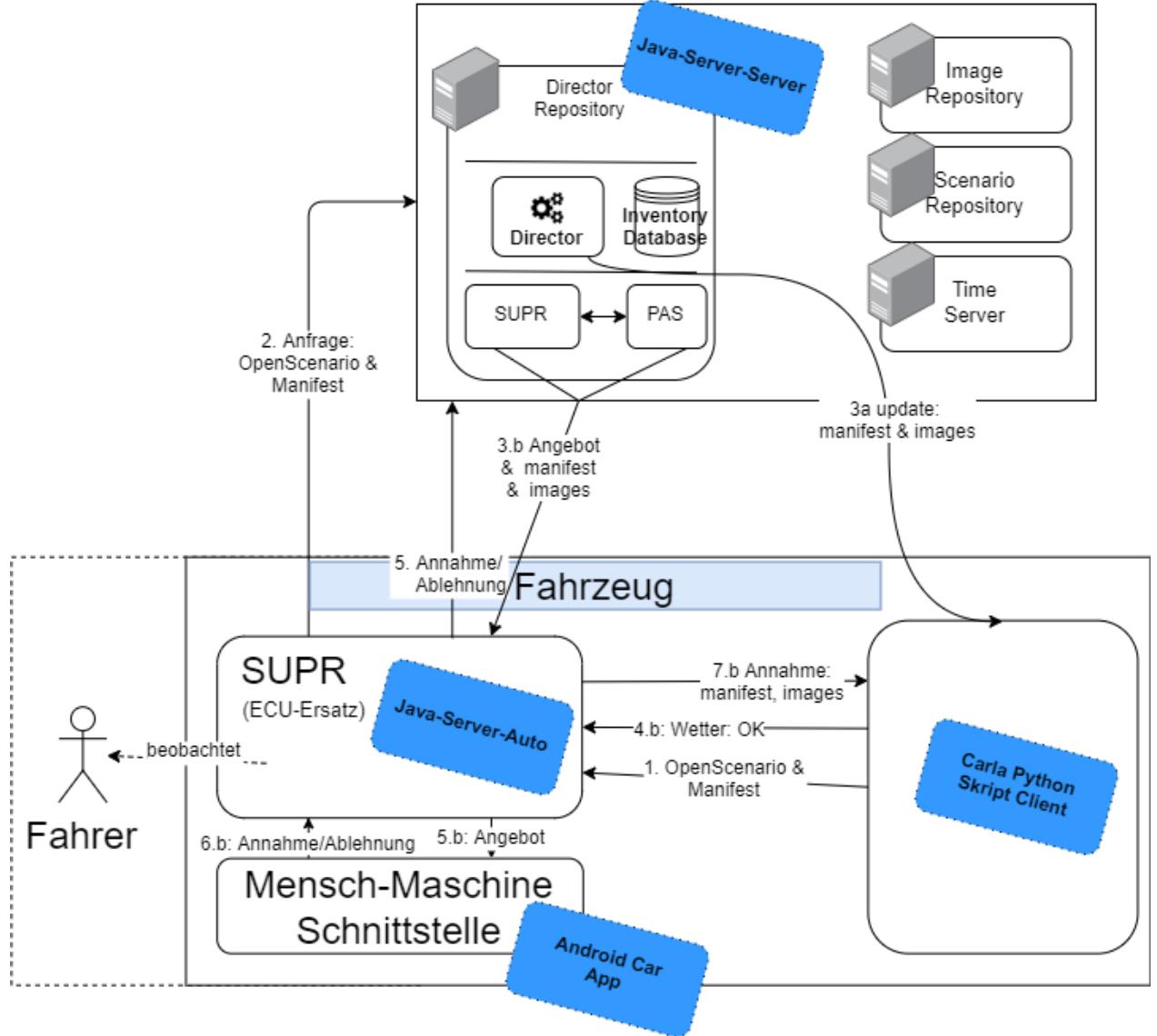


Abbildung 6: Technisches Erstkonzept

Die Architektur des Servers ist identisch zu der aus Kapitel 3.2. In der Implementierung sollen der Einfachheit halber die vier Bestandteile des Servers auf einem Server laufen und nicht getrennt werden. Der SUPR des Director Repositorys soll die im Forschungsseminar ihm zugeschriebenen Aufgaben durchführen können, gleiches gilt für den PAS. Die einzelnen Bestandteile des Servers können mit Ausnahme des Time-Servers miteinander kommunizieren.

Aus den Pfeilen des Konzepts abzuleiten ist der mögliche Ablauf eines Kaufprozesses. Initiiert wird dieser dadurch, dass das Fahrzeug im Carla Client nicht mehr autonom Fahren kann. Dieser erstellt infolgedessen eine OpenScenario-Datei und schickt diese gemeinsam mit dem Manifest, in welchem die bereits installierte Software notiert ist, an den SUPR (Pfeil 1). Dieser schickt den entdeckten Bedarf als Anfrage an den Ser-

ver (Pfeil 2), welcher infolgedessen überprüft welche Software den entdeckten Bedarf abdecken kann. Hierzu sucht der Director des Servers im Manifest des Fahrzeugs nach zu aktualisierender Software. Wird ein Defizit festgestellt, stellt der Server eine Verbindung mit dem Python Client her, welcher dieses Update installiert (Pfeil 3.a). Ist jede Software wieder aktuell stellt der SUPR des Servers fest, ob die Bedarfssituation mit den Updates bewältigt werden kann. Ist dies **nicht** der Fall, sucht der SUPR nach der passenden Software anhand der OpenScenario Datei. Ist ein Paket gefunden, wird das vom PAS erstellt Angebot zusammen mit dem unterschriebenen Manifest und dem Software-Image an den SUPR zurückgeschickt (Pfeil 3b).

Wird eine neue Software vorgeschlagen, hält der SUPR den Vorschlag so lange von der Nutzeroberfläche fern, wie der Fahrer sich noch in einer 'stressigen' Situation befindet. Ob dies der Fall ist, wird anhand des Wetters abgeleitet. Ist es regnerisch, nebelig oder ähnliches, soll das Angebot zurückgehalten werden. Bei sonnigem Wetter darf es dann auf der Nutzeroberfläche angezeigt werden. (Pfeil 4.b) Der Fahrer wird anhand einer Mitteilung darüber informiert, dass eine neue mögliche Software vorhanden ist. Der Fahrer kann das Angebot in der Nutzeroberfläche anpassen und es abschließend entweder annehmen oder ablehnen(Pfeil 6.b). Wird das Angebot angenommen, installiert der SUPR (Java-Client) die Software auf dem Fahrzeug, indem er eine Nachricht an das Python Skript schickt. Die Simulation wird erneut gestartet und das Auto kann die zuvor schwierige Situation bewältigen.

3 Das Business Model Canvas und Wertschöpfungskette

Damit die Verteilung von **Software** geeignet organisiert wird, bedarf es einer Plattform über welche Softwares orts- und zeitunabhängig Heruntergeladen werden können. Diese Plattform kann in Form eines Software Shops realisiert werden, über welchen Softwares von Fahrzeughaltern gekauft und installiert werden können. Der Shop stellt den Mittelsmann zwischen Fahrzeughaltern und den Softwareherstellern dar. Die installierten Softwares können unter anderem den Fahrumfang autonomer Fahrfunktionen erweitern oder das Auto mit anderen Akteuren des Straßenverkehrs verbinden. Ein bereitgestellter Shop sollte Automarkenübergreifenden entwickelt werden, da hierdurch zugleich ein großer Teil des Marktes gewonnen wird und außerdem einige Mehrwerte für Fahrzeughalter entstehen, wie die folgenden:

1. Nachhaltigkeit von Fahrzeugen

Nach dem verlassen des Fließbandes altert die Software eines Autos. Updates sind heutzutage nur beim Mechaniker möglich und ist zudem mit einem großen Aufwand verbunden. Durch eine Kabellose Schnittstelle sollen Fahrzeughalter gewünschte Software orts- und zeitunabhängig installieren können. Durch Softwares können Fahrzeuge möglicherweise sicherer (*weniger Unfälle*) und schonender (*geringere Abnutzung etc.*) gefahren werden, wodurch sich die Lebenszeit des Fahrzeugs verlängern könnte. Bleiben Fahrzeuge länger "*modern*", kann die Nachfrage an Neuwagen langfristig zurückgehen kann. Ist der Shop Automarken-übergreifend, können sämtliche Software des Shops für jeden Kunden weltweit verfügbar sein. Hierzu müssen sämtliche Fahrzeuge die Softwares installieren und komplizieren können.

2. Selbstständige Erweiterung des Fahrzeugs

Die schlechte Alternative zu einer kabellosen Bereitstellung von Software ist die stetige Fahrt zum Mechaniker um neue Softwares zu installieren. Hier erhält ein Fahrzeug eine festgelegte Sammlung an Softwares, der Fahrzeughalter hat also keine Auswahlmöglichkeiten. Hierdurch kann auf Dauer viel Software auf dem Fahrzeug installiert sein, die der Fahrzeughalter nicht benötigt. Durch einen Softwareshop wird es möglich, dass ein Fahrzeughalter nur die tatsächlich benötigte Software auf seinem Fahrzeug installiert kann. Des weiteren können die Kosten für Fahrzeughalter hierdurch skalierbar gehalten werden, da ein Fahrzeughalter selber entscheiden kann welche Softwares er kauft.

Um einen Überblick des Marktes zu geben, wird zunächst ein Business Model Canvas (*BMS*) erarbeitet. Anschließend werden relevante Bausteine der Wertschöpfungskette anhand der Erkenntnisse aus dem Forschungsseminar sowie des Business Models identifiziert und deren Aufgaben erläutert. Abschließend erfolgt eine Zusammenfassung des Kapitels und es werden die technischen Konzepte vorgestellt.

3.1 Business Model Canvas

Das 2004 von Alexander Osterwalder entwickelte Busines Model Canvas (*BMC*) schafft einen Überblick über die Aufgaben, die Kosten- und Partnerstrukturen sowie den Kundensegmenten eines Unternehmens. Es hilft den Fokus auf die wesentlichen Zielsetzungen dessen zu setzen.¹² Folgend werden die Kundensegmente, die Kundenbeziehungen, die Marketingkanäle und die Einnahmequellen betrachtet. Anhand dieser

¹²<https://ut11.net/de/blog/dein-geschaftsmodell-kompakt-der-business-model-canvas/>

werden anschließend die Nutzenversprechen sowie die Schlüsselressourcen und -aktivitäten bestimmt. Durch die abschließende Bestimmung von Schlüsselpartnern und der möglichen Kostenstruktur eines Softwareshops wurden alle wesentlichen Elemente eines Geschäftsmodells in ein skalierbares System gebracht." [28, S. 14]

3.1.1 Kundensegmente

Die Kundensegmente des Marktes lassen sich in Einzelkunden, Gruppen und Flottenbetreiber aufteilen. Flottenbetreiber lassen sich in zwei Segmente aufteilen: "*Leihe und Leasing*" umfasst Autovermieter, Unternehmen die ihren Mitarbeitern Leasingwagen bereitstellen, aber auch weitere wie Car-Sharing Unternehmen. Deren Kunden erwarten eine grundlegende Sammlung an Software im Mietfahrzeug vorzufinden und wollen selbstständig weitere Softwares auf eigene Rechnung installieren können. Neben Leihe und Leasing sind auch Unternehmen mit Firmenwagen ein gesondertes Kundensegment. Beide haben kleine oder große Fahrzeugflotten und müssen Softwarekäufe dementsprechend skalieren können.

Einzelkunden sind die übliche Autofahrer, die ein privates Fahrzeug besitzen und Software auf diesem installieren möchten. Gruppen sind mehrere Einzelkunden, die gemeinsam Software kaufen um hierdurch Kosten zu sparen. Hierdurch werden Familienkäufe aber auch Käufe mit Freunden möglich. Durch ihre enorme Größe ist es sinnvoll, auch dieses Segment weiter zu unterteilen.

- 18-25** Die 18-25 Jährigen sind mit modernen Technologien wie dem Computer und dem Smartphone aufgewachsen. Sie stellt die Gruppe mit dem durchschnittlich geringsten Einkommen dar. Für sie ist es intuitiv zum Handy, Computer oder anderen Alltags-unterstützenden Technologien zu greifen. Sie erkennen die potentiellen Mehrwerte von Technologien leichter als ältere Segmente und neigen daher vermutlich eher zum Kauf von Software.
- 25-33** Ebenfalls bestens mit Technik vertraut, umfasst dieses Segment die vermutlich wichtigsten Kunden. Es umfasst viele verdienende Menschen, die am Anfang ihrer Karriere stehen und dabei sind sich ein Leben aufzubauen. Sie sind in der Lage mehr Software als die jüngeren Segmente zu kaufen. Durch ihr fortgeschrittenes Alter sind sie für ältere Segmente oft ein Ansprechpartner im Bezug auf technologische Fragen. [30]
- 33-50** Fest im Leben stehend, stellt dieses Segment das Mengenmäßig größte dar. [19, S. 4] Im Gegenteil zu den jüngeren Segmenten ist hier wahrscheinlich dass die meisten ein eigenes Fahrzeug haben, auf welchem Sie Software installieren können.
- 50-65** Auch in diesem Segment haben die meisten ein eigenes Auto. [19, ebenda] Dieses wird sich öfters geteilt, da die Notwendigkeit für Zwei Autos nicht mehr gegeben ist. Die Anforderungen sind vergleichbar zu denen der 33-50 Jährigen, jedoch lässt sich diese Gruppe im Bezug auf neue Technik eher beraten.
- 65-75** Je älter der Kunde ist, desto geringer ist die durchschnittliche Technikaffinität. In diesem Segment sind die Mehrwerte von Technik maßgebend dafür, ob ein Kauf stattfindet oder nicht. Eröffnet sich Raum zur Kritik, neigen diese eher vom Kauf ab. [23] Zeitgleich sind sie von den jüngeren Segmenten einfach zu beeinflussen wenn es um den Kauf neuer Technik geht.
- 75 +** Durch das fortgeschrittene Alter benötigt dieses Segment Unterstützung bei der Autofahrt. Es legt Wert auf ein weitgehend selbstständig fahrendes Fahrzeug, da so Strecken zurückgelegt werden können die im Normalfall nicht bewältigt worden wären.

Sowohl Einzelkunden als auch Flottenbetreiber haben ähnliche Anforderungen an einen Software Shop. Durch neue Softwares soll das Fahrzeug vermehrt selbstständig fahren, um den Insassen Zeit zu ersparen. Kunden wollen die akquirierten Softwares verwalten und überwachen können, um so einen Überblick ihres Fahrzeugs zu haben.

3.1.2 Kundenbeziehungen

Obwohl das Einzelkundensegment quantitativ größer ist, haben alle Kundensegmente die gleiche Wichtigkeit. Um die Kunden nach dem ersten Kauf nicht zu Verlieren, muss eine positive Bindung zwischen ihnen und dem Shop aufgebaut werden. So sollte die erste Software, die dem Kunden vorgeschlagen wird einen deutlichen Mehrwert für diesen bieten. Hierdurch steigt die Zufriedenheit des Kunden und ein erneuter Kauf ist wahrscheinlicher.

Um langfristig viel Software über den Shop absetzen zu können, ist auch darüber hinaus eine gute Kundenbeziehung wichtig. Fahrzeughalter müssen **dem Shop vertrauen** können, bei **Kaufentscheidungen unterstützt** und **beraten** werden. Diese Ziele können erreicht werden, indem akquirierte Softwares deutliche Mehrwerte für Fahrzeughalter bieten. Auch die Einbeziehung von Fahrzeughaltern in die Entwicklung von Softwares kann eine positive Kundenbeziehung aufbauen. Um Flottenbetreiber beim Kauf von Software zu unterstützen, ist eine Web-App Sinnvoll, über die für eine große Menge an Fahrzeugen die gleichen Softwares gekauft, verwaltet und überwacht werden können.

3.1.3 Marketingkanäle

Um eine marktführende Position zu erreichen, sollten bei Markteintritt Werbekampagnen auf allen möglichen Marketingkanälen stattfinden. Die Altersgruppen des Einzelkundensegments werden über unterschiedliche Kanäle erreicht. Um ältere Segmente (50+) zu erreichen, ist das Nutzen *traditioneller* Medien wie Plakate, Zeitschrift oder dem Fernsehen Sinnvoll. Zu Sendezeiten, in denen überwiegend junge Zuschauer aktiv sind, sollen die Werbespots dementsprechend abgeändert werden. Jüngere Kunden sind aktiver auf anderen Medien oder Plattformen, wie Instagramm, Facebook, Twitter, aber auch und vor allem Youtube. Über diese kann zum einen Influencermarketing (*Influencer bewerben die Plattform*) betrieben werden oder auch klassische Werbeanzeigen geschaltet werden. Ziel ist es, dass viel über den neuen Shop geredet und berichtet wird, damit dieser allgegenwärtig wird.

3.1.4 Nutzenversprechen

1. Orts- und Zeitunabhängige Akquise eigens ausgewählter Softwares

Softwares sollen von Fahrzeughaltern orts- und zeitunabhängig heruntergeladen werden können. Die Softwares können selber ausgewählt werden. Hierdurch werden die Fahrten zum Mechaniker aufgrund von Softwareupdates gespart.

2. Überwachung, Verwaltung und Vorschlagen von Software

Fahrzeughalter sollen überwachen können, welche installierten Softwares wie oft genutzt werden. Hierdurch können nicht benötigte Softwares identifiziert und deinstalliert werden. Neben dieser Unterstützung, wird auch die Suche nach möglicherweise passenden Softwares für den Fahrzeughalter automatisiert. Dies kann

vor allem weniger technikaffine Kundensegmente beim Kauf unterstützen und Sicherheit im Umgang mit dem Fahrzeug schaffen. Wie der Bedarf einer Software erkannt werden kann, wird in Kapitel 2.2.3 erläutert.

3. Autofahren wird sicherer

Autonom fahrende Fahrzeuge sind in der Lage, den Straßenverkehr sicherer zu machen. [18] Neben den Sicherheitsvorteilen die einzelne Softwares für ein Fahrzeugs haben können, ist weitblickend vor allem der Einsatz von V2X-Kommunikation wichtig. Durch diese kann die Sicherheit des Straßenverkehrs entscheidend verbessert werden. [17, S. 19]

4. Lebenszeit des Autos wird verlängert

Aufgrund der steigenden Sicherheit werden Fahrzeuge künftig unfallfreier Fahren können. Außerdem wird die Fahraufgabe von einer Software schonender für Getriebe- und Motorteile durchgeführt. Durch den steigigen Kauf neuer Softwares und geregelten Updates bereits gekaufter Produkte kann die durchschnittliche Lebenszeit eines Fahrzeugs verlängern, was auch die Anschaffung von Neuwagen Fahrzeugs aufschieben kann.

5. Geringerer Wertverlust von Fahrzeugen

Durch das eigenständige erweitern des Fahrumfangs der Fahrzeugs kann der Wert steigen. Nicht nur steigert Preis der Software den Wert, sondern auch die bereits vorhandene Konfiguration der Softwares beinhaltet einen Wert an sich.

6. Fahrzeughalter gewinnen Zeit

Durch das Nutzen autonomer Fahrfunktionen können immer weitere Strecken zurückgelegt werden ohne dass der Fahrer selber die Steuerung übernehmen muss. Die hierdurch gewonnene Zeit kann zum Arbeiten oder auch anderweitig genutzt werden. Durch diese mögliche gemeinsame Zeit kann eine Autofahrt künftig weniger anstrengend sondern eher wertvoll sein.

7. Stetige Erweiterung des Softwareangebots

Die Bereitstellung eines offenen Markts für Softwareanbieter kann viele Entwickler anlocken, da der Umsatz auf diesem Markt sehr hoch sein könnte (*1 Million verkauft für 20€/Stück = 20 Millionen Umsatz*). Das Spektrum neuer Softwares kann durch die steigende Anzahl an Entwicklern zunehmend größer werden. Hierdurch können viele Anwendungsfälle schneller abgedeckt werden.

8. Interaktion mit der Autoumwelt

Bestimmte Softwares können die Kommunikation mit anderen Akteuren der Fahrzeugumwelt (*Ampeln, Parkplätze, andere Fahrzeuge, Drive-In-Restaurants, uvm.*) ermöglichen und so neue Möglichkeiten zur Interaktion untereinander schaffen.

3.1.5 Einnahmequellen

Der Softwareshop hat zwei unterschiedliche Einkommensströme. Zum einen kann es eine 'Anmeldegebühr' geben, die Softwareprovider zahlen müssen um Software im Shop veröffentlichen zu können. Außerdem können diese ihre Softwares im Shop bewerben, indem sie Werbeflächen kaufen wodurch ihre Software zum Kauf

hervorgehoben wird. Dies ist vergleichbar mit dem Ergebnis einer Google-Suche, bei der ganz oben *gesponserte* Internetseiten/Produkte zu finden sind.

Neben Einnahmen durch die Softwareprovider wird auch Umsatz durch den Verkauf und die Nutzung von Software generiert. Beim Kauf einer Software geht ein fester Prozentsatz des Verkaufspreises an den Shop Betreiber zurück. Zum Vergleich: Bei Android Apps liegt der Anteil den Google einnimmt bei 30%. Auch bei entstehenden In-App-Käufen (*z.B. für die Nutzung eines Services*) geht ein Anteil von 10% an Google.

Durch die Entwicklung von Entwicklungs- und Analyse-Tools können weitere Einnahmen generiert werden. Provider können diese kaufen und nutzen, wodurch sie bei der Entwicklung neuer Softwares unterstützt werden.

3.1.6 Schlüsselressourcen

Die wichtigsten Ressourcen sind die **geschaffene Plattform des Softwareshops** und die dort vertriebenen **Softwares**. Um eine große Menge an Softwares und sonstigen Daten von Fahrzeugen speichern und analysieren zu können, ist vor allem ein starkes und effizientes Backend wichtig, welches zugleich für die Suche und Verteilung von Software zuständig ist. Damit der Faktor der Ortsabhängigkeit von Softwaredownloads realisiert werden kann, ist ein stabiles Kommunikationsnetzwerk (z.B. 5G [17, S. 10]) notwendig. Softwares werden zum Großteil von Software Providern bereitgestellt. Ohne das stetige hinzufügen und aktualisieren von Softwares, verliert der Shop auf Dauer Attraktivität.

Eine weitere Schlüsselressource ist die Einbindung eines elektronischen Zahlungsservices, damit Fahrzeughalter Softwares tatsächlich bezahlen können.

3.1.7 Schlüsselaktivitäten

Die Schlüsselaktivitäten sind die Aufgaben des Unternehmens, die zur Erfüllung vorgestellter Nutzenversprechen führen. Sie sind an die Schlüsselressourcen und Nutzenversprechen gekoppelt und sollen die Anforderungen dieser erfüllen. Die Schlüsselaktivitäten stellen im Kontext einer Wertschöpfungskette **die wichtigsten Bausteine** dieser dar. Daher werden sie im folgenden zunächst in vier Gruppen unterteilt um in Kapitel 3.2 näher erläutert und in einen logischen Kontext gebracht zu werden.

1. Aktivitäten der Bedarfserkennung von Software

- Software Klassifizierung
- Automatische Erkennung möglichen Softwarebedarfs

2. Aktivitäten der Bereitstellung von Software

- Sicherheitsverifikation von Software anhand von Sicherheitskonzepten
- Sicherer Download über das Internet bereitstellen
- Bereitstellung einer elektronischen Zahlungsschnittstelle
- Angebotsunterbreitung

- Eigenständige Softwareentwicklung

3. Aktivitäten des generellen Betriebs

- Shop Verwaltung
- Shop (*Weiter-)Entwicklung und Wartung*
- Eigenständige Verwaltung von Software
- Eigenständige Überwachung von Software
- Kundenservice und individuelle Beratung

4. Aktivitäten zur Verbesserung der Supply Chain

- IDE(*Entwicklungsumgebung*) entwickeln
- Dokumentation & API Reference
- Tool Entwicklung (*Für Entwickler*)

3.1.8 Schlüsselpartner

Die Schlüsselpartner sind die Teilnehmer der Supply Chain, von denen der Software Shop direkt abhängig ist. Eine gute Beziehung zu Ihnen ist wichtig, um den Software Shop zu erhalten.

- **Software Provider**

Software Provider sind die Entwickler von Softwares. Durch ihre Gedankengänge werden vielfältige, individuelle Softwares möglich. Die Produkte von Software Providern dürfen **keine Sicherheitslücken enthalten**. Um dies zu unterstützen, müssen neue Entwickler den Aufbau einer Software für Fahrzeuge schnell verstehen und selber probieren können. Software Provider sind der wichtigste Partner des Software Shops und müssen auch dementsprechend beachtet und in den Entwicklungsprozess integriert werden.

- **Automobilhersteller**

Um mit dem Shop den größtmöglichen Erfolg zu erzielen, sollte dieser auf möglichst vielen Fahrzeugen vorhanden sein. Es ist daher wichtig, möglichst viele Automobilhersteller in die Supply Chain zu integrieren und somit den Markt zu erweitern. Automobilhersteller könnten andernfalls ein Konkurrenzprodukt veröffentlichen und mit diesem den möglichen Markt einschränken.

Der Software Shop sollte zudem auf jedem hergestellte Neuwagen vorinstalliert sein, um bei Verkauf des Fahrzeugs direkt verfügbar zu sein.

- **Mobilfunkbetreiber**

Da ein gut ausgebautes Mobilfunknetz wichtig für die Bereitstellung von Software ist, sollten Partnerschaften mit Mobilfunkbetreibern getroffen werden, damit Bandbreite für den Download von Software günstiger sein kann aber auch damit eine notwendige Netzwerk-Infrastruktur in einzelnen Ländern errichtet werden kann.

3.1.9 Kostenstruktur

Die Pflege der Beziehungen mit den Schlüsselpartnern ist wichtig, da der Software Shop ohne sie nicht existieren könnte. So ist die Entwicklung einer IDE, von Tutorials, API Referenzen und Analysetools für Software Provider zugleich wichtig aber auch kostspielig. Auch für sonstige Events des Unternehmens (*Konferenzen, Tagungen etc.*) sind hohe Kosten zu kalkulieren.

Es ist vorstellbar, dass Automobilhersteller prozentuale Gewinne des Shops erhalten. Diese Summe könnte anhand der Menge verkaufter Softwares auf Fahrzeugen dieser Marke berechnet werden. Weiterhin fallen Kosten für den Betrieb , die (Weiter-)Entwicklung und die Vermarktung des Shops an. Diese sind im wesentlichen für die Bezahlung von Managern, Entwicklern und Designern. Weiter fallen Stromkosten für das Betreiben von Servern an.

3.2 Die Supply Chain und die Wertschöpfungskette

Damit der Weg von Entwicklung, über Bereitstellung und letztendlichen Kauf nachvollzogen werden kann, werden die Schlüsselpartner in Abbildung 7 in einer Supply Chain dargestellt. Der Prozess startet bei Software Providern, welche eine Idee für eine Software haben und diese umsetzen. Hierzu ist möglicherweise die Zusammenarbeit mit Service Providern notwendig. Die entwickelte Software wird an den Software Shop weitergeleitet und dieser stellt sie entweder im Shop bereit oder nicht. Abschließend können Fahrzeughalter Software über eine kabellose Schnittstelle wie das Telefonnetz der Mobilfunkbetreiber Softwares herunterladen. Die Wertschöpfungskette ist als detailliertere Sicht des "Software Shops" (siehe Abb.) zu sehen. Sie stellt die Entscheidungen und Schritte dar, die eine Software von der Lieferung des Providers (**IN**) bis zum Download im Shop **OUT** durchlaufen und bestehen muss.

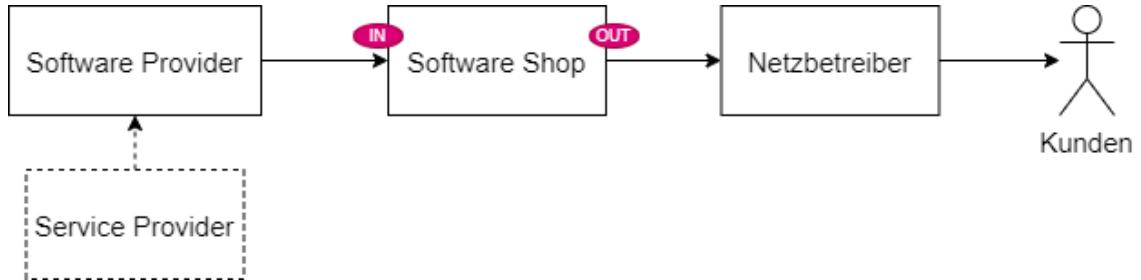


Abbildung 7: Supply Chain von Software

Nach Porter ist die Wertschöpfungskette in primäre und unterstützende Aktivitäten aufzuteilen. [21] Primäre Aktivitäten "liefern dabei einen direkten wertschöpfenden Beitrag zur Erstellung eines Produktes." [21, Vgl.] Unterstützende Aktivitäten sind als "notwendige Voraussetzung zu Erstellung der Produkte" [21, Vgl. ebd.] zu sehen. Sie sind im Rahmen aller Primäraktivitäten aktiv und beeinflussen die Qualität des Produktes. [24]. In Abbildung 8 werden die identifizierten Bausteine den Aktivitäten einer Wertschöpfungskette zugeordnet. Die in Kapitel 3.1.7 festgelegte Gruppierung der Schlüsselaktivitäten wird beibehalten und farblich hervorgehoben.

3.2.1 Primäre Aktivitäten

Es gibt fünf primäre Aktivitäten. Die erste Aktivität ist die Beschaffung und Lagerung von Materialien, in diesem Fall von Softwares, welche als **Eingangslogistik** bezeichnet wird. Im Kontext dieser sind zwei wichtige Bausteine zu erwähnen:

Software-Sicherheit verifizieren

Sämtliche Softwares die dem Shop hinzugefügt werden sollen, müssen zunächst diverse Sicherheitschecks überstehen. Zum einen sollten Softwares hinsichtlich aufgenommener und gespeicherter Daten überprüft werden. Je nach Land gelten unterschiedliche Datenschutzrichtlinien, welche zu beachten sind um die Privatsphäre von Fahrzeughaltern zu schützen. Weiterhin muss festgestellt werden, dass durch die neue Software keine Sicherheitslücken bei der Fahraufgabe entstehen. Möglicherweise kann dies durch die Nutzung von Simulationen erreicht werden.

Es ist sinnvoll, ein Sicherheitskonzept zu entwickeln welches die Überprüfung von Software unterstützt.

Das Ziel des Sicherheitsverifikation ist letzten Endes, dass durch neues Softwares keine Gefahren für die Fahrzeuginsassen entstehen.

Klassifizierung von Software

Erfüllt eine Software die Sicherheitsrichtlinien, wird sie anschließend Klassifiziert. Die Klassifizierung von Software soll die Suche und Darstellung dieser im Shop unterstützen, in dem der Software diverse Meta-Daten hinzugefügt werden. Teilweise können diese Meta-Daten im Shop als Such-Filter verwendet werden, einzelne geben auch an wie gut/schlecht eine Software ist und welche Aufgabe sie erfüllt. Ein beispielhaftes Konzept wird in Kapitel 4.1 erarbeitet und erklärt.

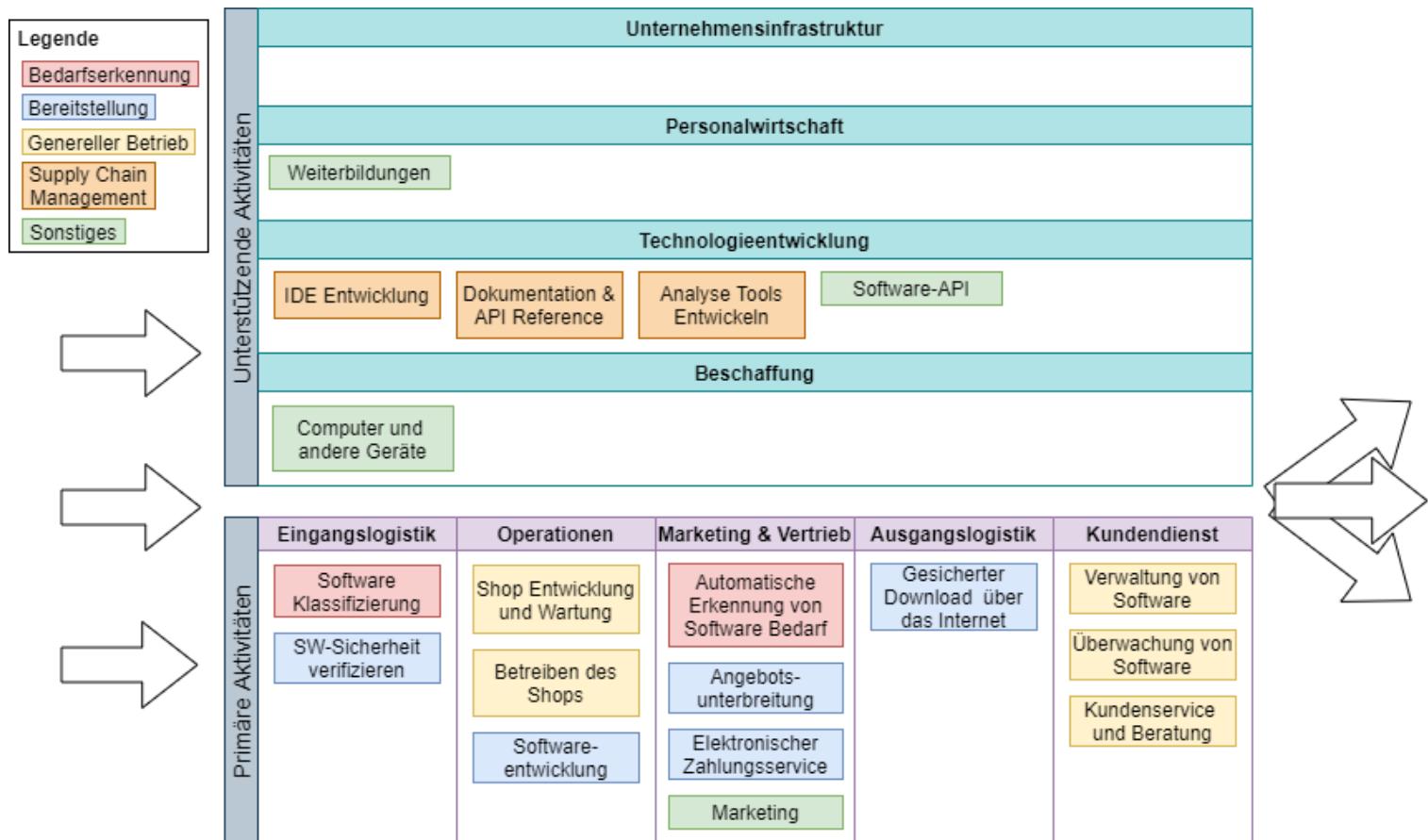


Abbildung 8: Die wichtigsten Bausteine einzelner Aktivitäten der Wertschöpfungskette

Nach erfolgreichen Durchlaufen der Eingangslogistik folgen die Aktivitäten der **Operation**. Im Rahmen dieser werden die verifizierten und klassifizierten Softwares dem Shop hinzugefügt und verwaltet. Die Operationen sind die Aufgaben des Unternehmens, die den eigentlichen Mehrwert für die Kunden schaffen.

Shop (Weiter-)Entwicklung und Wartung

Der Software Shop ist das Herz des Unternehmens, ohne welchen dessen Existenz nicht möglich ist. Die (Weiter-)Entwicklung und Wartung des Shops ist daher eine der Kernaufgaben des Unternehmens.

Neue Funktionen betreffen entweder das Front-End oder das Back-End des Shop. Das Front-End ist die grafische Oberfläche, die letzten Endes von den Fahrzeughaltern bedient wird. Bei der Front-End-Entwicklung kann die Verwendung des User-Centered-Designs zu einer Verbesserung von UI und UX (*User-Experience*) führen. Das Back-End ist für den Fahrzeughalter nicht sichtbar. Es **verarbeitet Kauf- und Suchanfragen von Software** und muss dementsprechend schnell und effizient sein, um eine große Menge derartiger Anfragen verarbeiten zu können. Da der Shop viel Software umfasst, die von Software Providern zur Verfügung gestellt wurde, sollte bei der (Weiter-)Entwicklung des Shops auf Kritik, Verbesserungsvorschläge und Wünsche dieser eingegangen werden. Auch die übrige *Community* (OEMs, Fahrzeughalter, Flottenbetreiber) sollte in die (Weiter-)Entwicklung des Shops einbezogen werden.

Neben der (Weiter-)Entwicklung des Shops, muss auch die bereits existierende Plattform gewartet werden. Dies umfasst den Austausch von unzuverlässiger Hardware, aber auch das schnelle Beheben von Bugs, die im Shop aufgetreten sind und die kontinuierliche Überwachung der Datenbanken, Server etc.

Betreiben des Shops

Damit der Shop betrieben werden kann, muss der Inhalt dessen verwaltet werden.

Softwares, die durch die Eingangslogistik klassifiziert und verifiziert wurden, müssen dem Shop hinzugefügt oder aus diesem entfernt werden und eingehende Download-Anfragen müssen verarbeitet werden. Softwares die vermehrt von Fahrzeughaltern gemeldet wurden, müssen überprüft und gegebenenfalls aus dem Shop entfernt werden. Dies kann der Fall sein, wenn eine Software Sicherheitslücken aufweist. Damit im Shop keine Inhalte (*Softwares, Bewertungen von Softwares*) vorkommen, die diskriminierend, rassistische oder auf eine andere Art & Weise Menschenrechts-verletzend sind, werden diese sofort aus dem Shop entfernt.

Softwares die im Laufe der Zeit als "besonders gutäufgefallen sind", können ein Siegel Empfehlung der Redaktion erhalten. Durch dieses Siegel soll die Qualität einer Software betont werden, wodurch Fahrzeughalter zum Kauf bewegt werden können.

Eigene Softwareentwicklung

Neben der Bereitstellung von Software anderer Software Provider ist auch die eigene Entwicklung und Bereitstellung von Software im Shop wichtig. Einerseits stellt der Verkauf eigener Software weitere Einnahmequelle für das Unternehmen, andererseits lässt der Verkauf einer guten, günstigen Software die Reputation des Shops steigen.

Die vorangegangenen Aufgaben stellen Voraussetzungen dar, welche für den Verkauf von Software notwendig sind. Der Shop ist zunächst auf Fahrzeugen installiert, aber Fahrzeughalter müssen noch zum Kauf bewegt werden. Dies passiert im dritten Teil der Wertschöpfungskette, dem **Marketing und Vertrieb** (von Software).

Automatische Erkennung von Softwarebedarf

Um Fahrzeughalter beim Kauf von Software zu unterstützen soll sichergestellt werden, dass die gekauften Softwares auch tatsächlich benötigt werden. Hierzu soll der Softwarebedarf eines Fahrzeug automatisch anhand von Fahrzeugdaten identifiziert und dem Fahrzeughalter vorgeschlagen werden. Es

gibt diverse Möglichkeiten den Softwarebedarf eines Fahrzeugs automatisch zu bestimmen:

Suche anhand einer einzelnen Situation

Diese Suche wird in Kapitel 2.2.3 beschrieben. Das Ergebnis der Suche entählt entweder **keine** oder genau **eine** Software. Die Suche soll automatisch erfolgen und die Vorschläge werden im Shop aufgezeigt.

Routen-Basierte Suche nach Software

Ebenfalls in Kapitel 2.2.3 beschrieben, erfolgt diese Suche vor Fahrtantritt und wird im übrigen ausgeführt, wenn das Fahrzeug kurz davor ist eine nicht häufig zurückgelegte Strecke zu Fahren. Die Ergebnisse der Suche werden in die Routenplanung integriert und dem Fahrzeughalter wie in Kapitel 2.2.3 beschrieben vorgeschlagen.

Shop-Basierte Suche (*Shop-Auswahl*)

Wird eine Software unabhängig von vorherigen Suchen im Shop ausgewählt, kann der Fahrer berechnen lassen, ob und wenn ja zu welchem Umfang die jeweilige Software das Fahrspktrum des Fahrzeugs erweitert. Das Ergebnis der Anfrage stellt eine gute Entscheidungsgrundlage für den Kauf der Software dar, da dem Fahrzeughalter so der tatsächliche eigene Nutzen der Software präsentiert wird.

Umgebungsbedingte Softwaresuche

Die Umgebungssuche ist eine Weiterentwicklung der in Kapitel 2.2.3 erwähnten *Suche aus Basis der Fahrtenhistorie*. Die Idee hinter der Umgebungs-Suche ist, dass dem Fahrzeughalter Softwares vorgeschlagen werden, die andere Fahrzeuge in der Region vermehrt installiert haben. Die Umgebungs-Suche wird in Kapitel 4.2 nochmals im Detail erläutert.

Abhängig davon, ob passende Softwares gefunden wurden oder nicht, werden sie dem Fahrzeughalter vorgeschlagen. Durch den Prozess wird nicht nur die Kundenzufriedenheit gesteigert, sondern auch mehrere Nutzenversprechen erfüllt(NV-3, NV-4, NV-5). Neben Suchalgorithmen die dem Fahrzeughalter Softwarevorschläge unterbreiten, können Service Provider einem Fahrzeug auch eine Software vorschlagen. Dies ist notwendig, wenn der Fahrzeughalter einen Service nutzen will für den eine bestimmte Software auf dem Fahrzeug installiert sein muss.

Angebotsunterbreitung

Wird anhand einer der zuvor aufgelisteten SUCHEn festgestellt, das eine bestimmte Software dem Fahrzeug hinzugefügt werden kann, muss diese dem Fahrzeughalter vorgeschlagen werden. Im Kontext der Angebotsunterbreitung wird der Zeitpunkt festgestellt, zu welchem die Software dem Fahrer vorgeschlagen wird. Ein Konzept für diese Situationserkennung wurde in Kapitel 2.3.3 erläutert.

Bereitstellung elektronischer Zahlungsschnittstellen

Um den Kauf einer Software abschließen zu können, müssen Fahrzeughalter den Kaufpreis über eine elektronische Zahlungsschnittstelle bezahlen können. Hierzu sollten bestenfalls mehrere gängige Zahlungsschnittstellen (*MasterCard, PayPal, EC-Cash o.A.*) mit dem Shop verknüpft werden, um so mehr Fahrzeughaltern den Kauf zu ermöglichen. Um weitere eigene Umsätze zu generieren, kann hier auch eine eigene Zahlungsschnittstelle hinzugefügt werden. Fahrzeughalter können zur Verwendung dieser gebracht werden, indem der Kauf mittels dieser Schnittstelle beispielsweise günstiger oder schneller ist.

Marketing

Wie im Business Model beschrieben, ist der Umfang des Marketings maßgebend für den Erfolg des Shops. Die Kampagnen sollten auf möglichst vielen Kanälen zu sehen sein und dabei möglichst viele Alters- und Kundengruppen ansprechen. Vor allem die Vermarktung über Social Media Kanäle ist ratsam, da über diese vorwiegend junge Menschen erreicht werden. Es kann Sinnvoll sein "Markenbotschafter*innen zu finden, die Content für Social Media Plattformen entwerfen. Coca Cola hat dies beispielsweise erfolgreich mit Coke TV¹³ geschafft.

Damit die beworbenen und vorgeschlagenen Softwares letztendlich auch an die Fahrzeuge "ausgeliefert" werden können, muss die **Ausgangslogistik** strukturiert werden. Sie komplettiert die Logistik *Eingangs- und Ausgangslogistik* und legt fest, über welche Distributionskanäle Software verteilt wird.

Gesicherter Download über das Internet

Die Bereitstellung von der bezahlten Software wird mit dem Download abgeschlossen. Dieser Download soll zum einen orts- und zeitunabhängig sein, außerdem muss die Download-Schnittstelle sicher sein. Um letzteres zu gewährleisten, sollte der Ausbau von Kommunikationsnetzwerken (*bspw. 5G*) gefördert werden. Durch den Download von Software über das Internet können hohe Kosten für Fahrzeughalter entstehen. Durch eine Kooperation mit Mobilfunkanbietern können diese Kosten skalierbar gehalten werden, was fortlaufend die Kundenzufriedenheit sichern kann.

Damit die langfristige Zufriedenheit von Kunden sichergestellt werden kann, müssen auch nach dem Kauf für den Kunden wertschöpfende Aktivitäten durchgeführt werden. Diese werden dem letzten Schritt der Wertschöpfungskette, dem **Kundendienst**, zugeordnet.

Verwaltung von gekaufter Software

Damit Fahrzeughalter wissen, welche Softwares sie gekauft, geliehen oder gemietet haben soll für sie die eigenständige Verwaltung von Software im Shop möglich sein. Fahrzeughalter behalten hierdurch den Überblick über ihr Fahrzeug und können die Softwares entfernen oder hinzufügen.

Überwachung installierter Software

Um Fahrzeughalter vor Augen zu führen, welche Softwares notwendig für das Fahrzeug sind und welche nicht, sollten diese Einblick in eine Art Überwachung der Software erhalten. Hier können grundlegende Statistiken geführt werden wie

- die durchschnittliche Nutzungsdauer der Software in Prozent
- das Datum, an welchem die Software das letzte mal genutzt wurde
- oder eine Liste, in welcher zu sehen ist welche Softwares an welchem Tag der Woche vermehrt genutzt werden.

Aufgrund der steigenden Bedeutung des Datenschutzes könnte auch eine Ansicht erstellt werden, welche zeigt welche Daten von Softwares erhoben werden. Hierdurch könnte das Vertrauen von Kunden gestärkt werden (*Oder auch nicht, je nach dem welche und wie viele Daten erhoben und gespeichert werden*).

Kundenservice und Beratung

Um vor allem die Bindung zu älteren Kunden aufzubauen, kann das einrichten einer Telefonischen

¹³https://www.youtube.com/channel/UCTfwo-EWSD-8hZ7F8HG_qJg, Aufgerufen am 14. Juni 2020

Beratung für sinnvoll sein. Vor allem bei der Ersteinrichtung eines Fahrzeugs können Fahrzeughalter unterstützt werden, indem sie telefonisch beraten werden. Auch alternative, vorwiegend digitale Varianten Fahrzeughalter bezüglich Sofwtares zu beraten kann einen Mehrwert für diese bieten. Durch das entwickeln von Nutzerforen oder der Möglichkeit, Softwares zu bewerten (*1-5 Sterne + Kommentar*) können Fahrzeughalter sich gegenseitig bezüglich Softwares beraten.

Überblick

Die fünf Aktivitäten der Wertschöpfungskette schaffen Mehrwerte für Fahrzeughalter. Wird eine Software von einem Software Providern beim Softwareshop eingereicht", wird sie für den Verkauf an Fahrzeughalter vorbereitet. Jeder einzelne Schritt der Wertschöpfungskette trägt dazu bei, das Fahrzeughalter am ende selbstständig genau die Software installieren können, die sie benötigen und dabei zufrieden mit dem Ergebnis sind. Während all diese Schritte durchlaufen werden, fügen die *unterstützenden Aktivitäten* stetig Mehrwerte zur Wertschöpfung hinzu und sind daher *indirekt am Erfolg beteiligt*. Im folgenden diese vorgestellt und ihre Rolle im Ablauf der Wertschöpfungskette verdeutlicht.

3.2.2 Unterstützende Aktivitäten

Es gibt vier Unterstützende Aktivitäten. Angefangen bei der Schaffung einer gut aufgebauten **Unternehmensinfrastruktur**, kann diese Alltägliche Problematiken beheben und zeit sparen. Zu dieser gehört ein gute Anbindung zur Außenwelt, schnelles Internet und Mitarbeiter freundliche Büroräume, in welchen auch Freizeitaktivitäten möglich sind. Neben der Schaffung einer guten Unternehmensinfrastruktur hat auch die **Personalwirtschaft** (*Human-Resource-Management*) einen starken Einfluss auf den Erfolg des Unternehmens. Die Personalwirtschaft beschreibt wichtige, die Mitarbeiter unterstützende Strukturen und Regelungen des Unternehmens.

Weiterbildungen

Die kontinuierliche Weiterbildung der Mitarbeiter ist für den Erfolg des Shops überaus wichtig. Weiterbildungen müssen nicht Zwangsweise in dem jeweiligen Fachbereich des Mitarbeiters liegen, sondern können auch neue Aspekte beleuchten. So können Entwickler einerseits gemeinsamen neue Programmiersprachen & -frameworks lernen, anderseits können auch andere SSkills" beigebracht oder Aktivitäten (*bspw. Sport*) miteinander durchgeführt werden, welche die Team-Chemie steigern. Die Schaffung einer guten Work-Life-Balance kann die Produktivität der Mitarbeiter erhöhen [16] und somit auch den Erfolg des Shops nachhaltig sichern.

Für ein digitales Unternehmen ist die stetige **Technologieentwicklung** wichtig und kann wegweisend für den Erfolg des Unternehmens sein. Zu dieser gehört nicht nur die Schaffung von Werten für die innerbetriebliche Wertschöpfungskette, sondern auch die Supply Chain übergreifend Unterstützung der Partner und Kunden.

IDE Entwicklung

Damit Software Provider bei der Entwicklung von Softwares unterstützt werden, ist die Bereitstellung einer eigens entwickelten Entwicklungsumgebung(*IDE*) sinnvoll. Diese IDE kann Features und Tools umfassen, welche explizit für die Entwicklung von (Fahrzeug-) Softwares entwickelt wurden. Durch integrierte Simulatoren und Sicherheitschecks könnte der Weg von Idee über Entwicklung hinzu Bereitstellung im Shop verkürzt werden. Ein Vergleich aus der Realität ist Android Studio¹⁴ aber auch die

¹⁴<https://developer.android.com/studio>

Eclipse IDE, welche in Zusammenarbeit mit Unternehmen aus der ganzen Welt entwickelt wird.¹⁵

Dokumentation und API-Reference

Die Entwicklung von Software wird zusätzlich durch die Bereitstellung einer ausführlichen API-Reference sowie technischen Dokumentation (mit Beispielen) positiv unterstützt. Die API-Reference sollte alle Funktionen von Klassen ausführlich beschreiben und im Nutzungskontext vorgestellt werden. Diagramme unterstützen zusätzlich das Verständnis der API. Um die Entwicklung neuer Softwares voranzutreiben ist es sinnvoll eine Einführung (*'Getting Started Guide'*) in die Entwicklung von Softwares für ein Fahrzeug bereitzustellen. Google stellt in diesem Kontext das *Android Jetpack*¹⁶ sowie weitere Dokumentationen mit Anleitungen und Videos zur Verfügung.¹⁷ Auch und vor allem sollten hier die Sicherheitskriterien welche im Kontext der Wertschöpfungskette in Abschnitt **Eingangslogistik** erwähnt wurden vorgestellt werden.

Entwicklung von Analyse-Tools

Um Software Provider zusätzlich bei der (Weiter-)Entwicklung zu unterstützen, können Analyse-Tools entwickelt werden. Anhand dieser können Statistiken von Softwares erhoben werden, durch welche Entwicklern Ideen zur Verbesserung ihrer Software leichter finden können. Dies Statistiken können generelle Kennzahlen (*Nutzung in H je Tag*) aber auch spezifischere Werte (*Button-Clicks*) sein, durch welche Entwickler dazu in der Lage sind, ihre Software zu verbessern.

API-Entwicklung

Damit Software Provider bestimmte Systeme von Fahrzeugen ansteuern können, müssen APIs für die jeweiligen Systeme des Fahrzeugs entwickelt werden. Über diese können Daten der Systeme ausgelesen oder eingefügt werden. Da künftig hergestellte Fahrzeuge neuartige Technologien enthalten werden, die heutzutage noch nicht existieren, müssen die APIs im Laufe der Zeit immer weiter angepasst bzw. erweitert werden.

Damit alte "Fahrzeuge weiterhin von einer Software gesteuert werden können, müssen die alten APIs aber bestehen bleiben. In Android wird diese Problematik sinnvoll durch das *ÄPI-Level*" gelöst: Jede Android Version hat ein eigenes *API-Level*. Je höher das Level, desto höher auch die Android Version. Ein Smartphone mit API-Level 23 (*Android 6.0*) kann alle Funktionen und Methoden "*tieferer*" APIs nutzen, nicht aber die "*höherer*" APIs (>23). Jede App hat eine *minimale API-Version* welche festlegt, welches API-Level das Endgerät mindestens haben muss. Im Kontext von Fahrzeugen ist es vorstellbar, dass jedes Teilsystem eines Fahrzeugs ein eigenes minimales API-Level hat. Softwares dadurch nur auf Fahrzeugen installiert werden, welche den Anforderungen der Software entsprechen.

Die letzte unterstützende Aktivität ist die **Beschaffung**. Ihre Aufgabe ist, dass alle Mitarbeiter mit den nötigen Materialien, das heißt Computern, Tastaturen, Stiften oder anderem versorgt werden.

¹⁵<https://www.eclipse.org/org/>, Aufgerufen am 14. Juni 2020

¹⁶<https://developer.android.com/jetpack>, Aufgerufen am 14. Juni 2020

¹⁷<https://developer.android.com/guide>, Aufgerufen am 14. Juni 2020

3.3 Zusammenfassung

Das in Kapitel 3.1 erstellte Geschäftsmodell bietet einen guten Überblick das Software Shops und die damit einhergehenden Aufgaben, Kosten, Partner und Strukturen der Unternehmung. Anhand der bestimmten Kundensegmente konnten Kundenbeziehungen definiert und Aspekte des Marketings identifiziert werden. Weiterhin wurden Nutzenversprechen aufgestellt, anhand welcher die Schlüsselressourcen, -aktivitäten und -partner identifiziert wurden.

In Kapitel 3.2 wurden die zuvor im Geschäftsmodell identifizierten Schlüsselaktivitäten sortiert und den einzelnen Aktivitäten einer Wertschöpfungskette nach Porter zugeordnet. Durch die weitere Aufteilung der Bausteine in fünf Gruppen (*Bedarfserkennung, Bereitstellung, Kundenzufriedenheit, Supply Chain Management, Sonstiges*) konnte verdeutlicht werden, welchen Sinn einzelne Bausteinen haben steckt und worauf diese Einfluss nehmen.

Um den Ablauf eines Softwareeinkaufs im Prototypen darzustellen, werden im folgenden technische Konzepte vorgestellt die in diesem implementiert werden können.

4 Technische Konzepte

Im vorherigen Kapitel wurden viele mögliche Aufgaben benannt, die im Kontext eines Software Shops entstehen können. Da in dieser Bachelorarbeit nicht alle identifizierten Bausteine im Detail diskutiert werden können, sollen die folgenden technischen Konzepte zumindest einige dieser erweitern. Im kommenden Kapitel wird ein mögliches Konzept für die Klassifizierung von Software vorgestellt, welche in der Eingangslogistik stattfindet. In Kapitel 4.2 wird ein Such-Algorithmus skizziert, welcher für die automatische Erkennung von Softwarebedarf genutzt werden kann. Im letzten Kapitel werden drei Kommunikationskonzepte vorgestellt, welche den Ablauf der Kommunikation zwischen dem Fahrzeug, dem Server und den anderen Akteuren der Umwelt im Kontext bestimmter Anwendungsfälle skizzieren.

4.1 Klassifizierung von Software

Das Ziel der Software Klassifikation ist es, den Fahrzeughaltern beim Kauf unterstützende Kennzahlen zur Verfügung zu stellen, anhand welcher sie sinnvollere Kaufentscheidungen treffen können und somit potentiell einen größeren Wert von der gekauften Software erhalten.

4.1.1 Berechtigungen

Eine dieser Kennzahlen ist die Anzahl genutzter Berechtigungen des Fahrzeugs. Nicht nur die Anzahl, sondern auch der Umfang einer Berechtigung können Kunden vom Kauf abschrecken. [22, S. 107] Eine Berechtigung erteilt einer Software die Erlaubnis, auf einzelne Systeme des Fahrzeugs zugreifen zu dürfen. Berechtigungen werden von Entwicklern zu einer Software hinzugefügt, wodurch diese im Shop für Fahrzeughalter zu sehen sind. Der Fahrzeughalter muss den Berechtigungen beim Kauf einer Software zustimmen, damit diese in vollem Umfang installiert und genutzt werden kann. Stimmen Fahrzeughalter Berechtigungen nicht zu, kann die entsprechende Software nicht gekauft werden. Durch die Verwendung von Berechtigungen kann die Sicherheit von Softwares gesteigert werden da sie verhindern, dass eine Software ohne weiteres auf alle Systeme des Fahrzeugs zugreifen kann. Im Kontext autonomer Fahrzeuge können mögliche Berechtigungen von der Architektur dieser abgeleitet werden, welche in Abbildung 9 dargestellt wird.

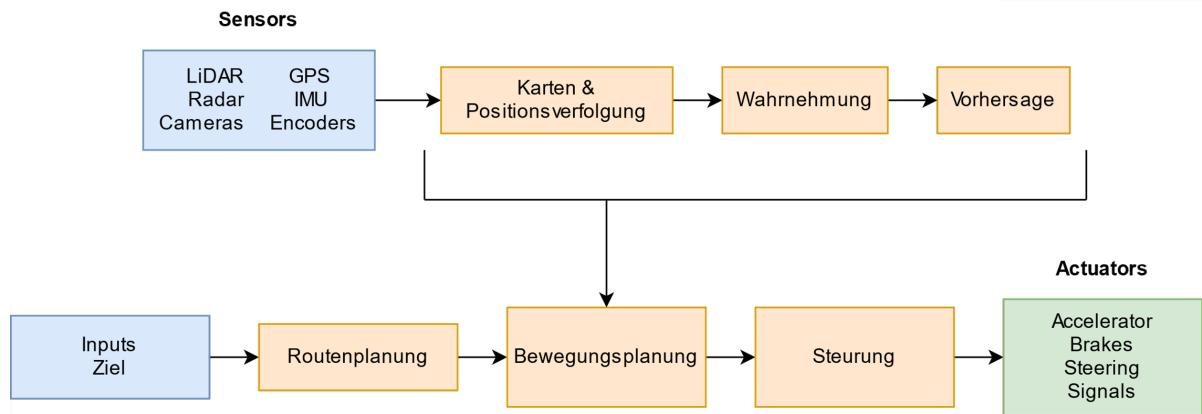


Abbildung 9: Jeff Schneider: Architecture of Autonomous Vehicles

Damit Berechtigungen die Fahrzeughalter davor schützen, persönliche Informationen unfreiwillig zu teilen und einer Software zu viel "Macht" über das Fahrzeug zu geben, sollte der Umfang von Berechtigungen möglichst klein gehalten werden. So sollte jede Sensorgruppe des Fahrzeugs eigene Berechtigungen haben, da die Art der aufgenommenen Informationen sich voneinander unterscheidet. So kann durch das auslesen des GPS-Sensors kontinuierlich die Position des Fahrzeugs aufgezeichnet werden, während dies über den Radar nicht möglich ist. Würde alle Sensoren über die gleiche Berechtigung zugänglich sein, würde dies eine Sicherheitslücke darstellen.

Neben der Privatsphäre der Fahrzeughalter ist vor allem die Sicherheit von Fahrzeuginsassen wichtig. Wird eine Software entwickelt, welche die Wahrnehmung, Vorhersage, Zieleingabe, Routenplanung, Bewegungsplanung, die Steuerung oder einzelne Aktuatoren beeinflusst, kann dies die Sicherheit der Insassen gefährden. Softwares, die diese Systeme nur auslesen aber nicht direkt beeinflussen haben keine Auswirkung auf die Fahrsicherheit. Daher sollte jedes System jeweils zwei einzelne Berechtigungen haben. Durch die eine kann ein System ausgelesen werden. Durch die andere können Systeme beeinflusst werden, wodurch aktiv in die Fahraufgabe eingegriffen wird.

Neben Berechtigungen, die einzelne Systeme eines Fahrzeugs benötigen, sollten auch weitere Services des Fahrzeugs nur durch die Nutzung von Berechtigungen möglich sein. Eine Berechtigung für die Nutzung des Internets ist sinnvoll, da Softwares hierdurch nicht wahllos Nutzerdaten speichern können oder sonstige Verletzungen von Datenschutz- oder sonstigen Sicherheitsrichtlinien begehen können. Auch die Kommunikation mit anderen Akteuren des Straßenverkehrs sollte über eine Berechtigung beschränkt werden.

4.1.2 Abgeleitete Kennzahlen

Neben der Art und Anzahl genutzter Berechtigungen einer Software, können auch andere Kennzahlen einer Software den Fahrzeughalter bei der Kaufentscheidung unterstützen. Einige dieser Kennzahlen können von den im Businessmodel beschriebenen Nutzenversprechen abgeleitet werden (*Kapitel 3.1.4*).

Um darzustellen, dass die Lebenszeit des Autos verlängert wird (*NV-4*) kann ein Wert berechnet werden, der aussagt wie gut eine Software die einzelnen Autoteile (*Motor, Gertiebe, Reifen etc.*) schont. Die Berechnung dieses Werts kann in Carla erfolgen. Zunächst müssen zwei Fahrzeuge simuliert werden, von denen das eine die jeweilige Software installiert hat und das andere nicht. Beide Fahrzeuge sollen die Szenarien, welche der Software beigefügt wurden, durchfahren und dabei bestimmte Werte wie die Reifenreibung, die Dämpfungsrate oder andere messen. Durch einen Vergleich der gemessenen Werte kann prozentual angegeben werden, wie stark eine Software bestimmte Teile des Fahrzeugs schont **oder auch eben nicht**. Um Fahrzeughaltern vor Augen zu führen wie viel Zeit durch eine Software gewonnen werden kann (*NV-6*), sollte ein Zeitwert berechnet werden, welcher die durchschnittlich autonom zurückgelegte Zeit einer Software bestimmt. Die Berechnung sollte auf Basis realer Nutzungsdaten durchgeführt werden. Neben der durchschnittlich autonom zurückgelegten Zeit einer Software kann auch die Anzahl an Interaktionen mit der Autoumwelt ein ausschlaggebender Faktor für den Kauf einer Software sein.

Angenommen eine Software soll das Fahren in der Innenstadt ermöglichen. Wohnt der Fahrzeughalter nicht im Umkreis einer Stadt, würde ihm die Software nichts sonderlich viel bringen. Durch die hohe Anzahl

gesparter Stunden ist der Fahrzeughalter überzeugt und kauft sich diese dennoch. Um derartige Fälle zu verhindern, sollt die Berechnungen von Kennzahlen an die jeweilige Region des Fahrzeugs angepasst sein. Weiter Kennzahlen können sein:

- 1. Anzahl an Downloads und Deinstallationen**
- 2. Benötigter Speicherplatz der Softwrae**
- 3. Bewertung der Software durch Fahrzeughalter**

Die vorgestellten Softwareberechtigungen und -kennzahlen werden im Shop zu jeder Software angezeigt, wodurch Fahrzeughalter einen ersten Eindruck einer Software erhalten können. Neben der Darstellung im Shop können Kennzahlen und Berechtigungen auch als Such-Filter im Shop verwendet werden. Hierdurch wird die Suche nach bestimmter Software im Shop erleichtert und Fahrzeughalter können leichter eine für sie passende Software entdecken.

4.2 Umgebungsbedingte Softwaresuche

In Kapitel 3.2 wurden mehrere Möglichkeiten aufgezählt, mit denen der Softwarebedarf eines Fahrzeugs automatisch bestimmt werden kann. Die im folgenden vorgestellte Umgebungssuche soll einem Fahrzeughalter Softwares vorschlagen, welche in der Umgebung des Fahrzeugs des öfteren von anderen Verkehrsteilnehmern gekauft oder genutzt wurden.

Die Umgebungssuche beginnt mit dem Verschicken einer Nachricht vom Fahrzeug an den Server. In dieser Nachricht ist das Manifest des Fahrzeugs enthalten, auf welchem unter anderem eine Liste aller installierter Softwares eines Fahrzeugs zu finden ist. Außerdem werden Fahrdaten verschickt, anhand welcher eine "Heatmap" erstellt werden kann durch welche ersichtlich wird, in welchen Regionen sich das Fahrzeug zuletzt aufgehalten hat. Der Server sucht innerhalb dieser Regionen nach Softwares, die von anderen Fahrzeugen in diesen gekauft wurden und noch nicht auf dem anfragenden Fahrzeug installiert sind. Ist die Suche abgeschlossen wird geprüft, ob die gefundenen Softwares einen Mehrwert für das Fahrzeug bieten. Um dies zu bestimmen, können die einer Software beigelegten Open Scenario Dateien genutzt werden, indem diese in einer Simulation durchfahren werden.

Zunächst wird eine virtuelle Instanz des Fahrzeugs anhand des versendeten Manifests erstellt (*Originalinstanz*). Für jede gefundene Software wird eine weitere Instanz des Fahrzeugs erstellt, auf welcher zusätzlich die Software installiert wird (*Neue Instanz*). Nach dem erstellen der virtuellen Fahrzeuge, durchlaufen die Originalinstanz und die neue Instanz die jeweiligen Szenarien der neuen Softwares. Durch einen Vergleich der Simulationen soll festgestellt werden, ob eine Software positive, negative oder überhaupt keine Auswirkungen auf das Fahrverhalten des Fahrzeugs hat. Positiv ist, wenn das Fahrzeug die Situation nun selbstständig durchfahren kann oder dies Ressourcenschonender tut. Negativ ist es, wenn die Situation durch die neue Software nicht weiter durchfahren wird oder die neue Software zum durchfahren der Situation nicht vom Fahrzeug ausgewählt wird. Dies kann bedeuten, dass auf dem Fahrzeug bereits eine Software installiert ist, welche diese bereits abdeckt.

4.3 Kommunikationsprotokolle

Damit Software heruntergeladen und anschließend genutzt werden kann, müssen das Fahrzeug, der Software Shop und auch potentiell weitere Akteure des Straßenverkehr (*Autos, Ampeln, Service Provider, o.A.*) miteinander kommunizieren. Damit diese Kommunikation sicher und geordnet ist, werden drei Kommunikationsprotokolle erstellt die im Kapitel 5 implementiert werden können. Die Kommunikation zwischen dem Fahrzeug und dem Server kann durch das einbinden der Uptane-Architektur (2.3.1) und einhergehender Lösungsansätze sicherer gestaltet werden. So enthält jede Nachricht die ein Fahrzeug an den Server schickt das jeweilige Manifest des Fahrzeugs, durch welches der Server das Fahrzeug verifizieren kann und diesem anschließend neue Softwares hinzufügen kann.

4.3.1 Kommunikationsprotokoll: Selbstständiger Softwarekauf

Das in Abbildung 10 dargestellte Protokoll stellt den Kommunikationsablauf beim Kauf einer spezifischen Software dar. Der Fahrzeughalter kann im Shop nach Softwares suchen, indem der Name einer Software gesucht wird oder Suchfilter verwendet werden. Wurde eine Software ausgewählt (1), wird eine Installationsanfrage für diese an den Server geschickt. Kann die Software auf dem Fahrzeug installiert werden, bereitet der Server ein **Installationspaket** vor (2). In diesem ist die Software, eine neue Version des Manifests und das Angebot für die Software enthalten. Ist das Installationspaket vorbereitet, wird es an das Fahrzeug verschickt und der Fahrzeughalter wird über die Mensch Maschine Schnittstelle über darüber informiert (3). Abschließend wird die Software auf dem Fahrzeug installiert und der Fahrzeughalter kann sie nutzen.

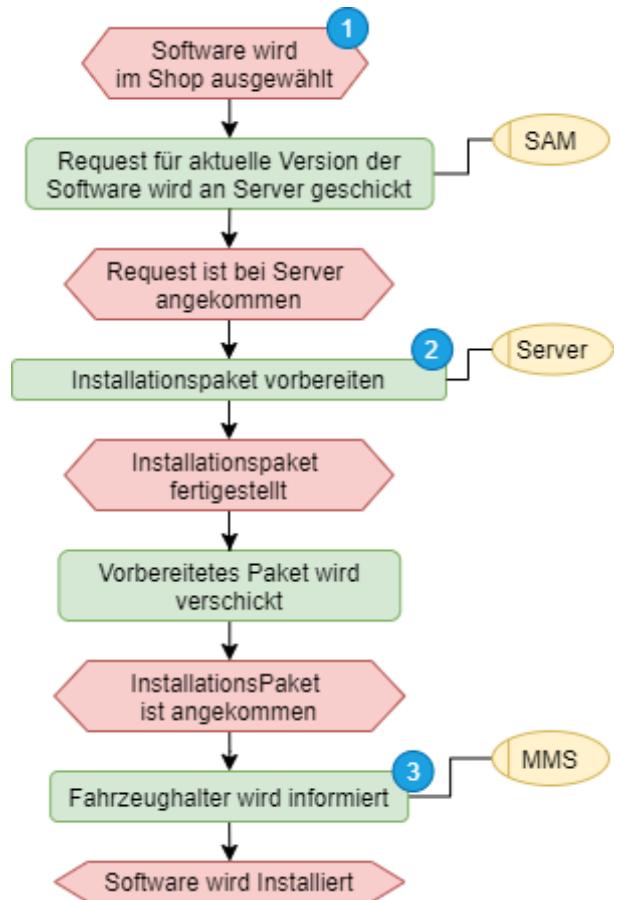


Abbildung 10: Eigenständige Installation von Software

4.3.2 Kommunikationsprotokoll: Installationsvorschlag vom Service Provider

In Abbildung 11 wird dargestellt, wie ein Service Provider einem Fahrzeug eine Software vorschlagen kann. Die Notwendigkeit hierfür besteht, da zur Nutzen eines Services jeweils eine bestimmte Software notwendig ist. Vorzustellen sei folgende Situation: Ein Fahrzeughalter möchte in der Oldenburger Innenstadt einkaufen gehen und muss hierfür sein Fahrzeug parken. Über das Navigationssystem wählt er einen Parkplatz aus den das Fahrzeug daraufhin ansteuert. Um ein Parkticket für diesen Parkplatz kaufen und das Fahrzeug automatisch einparken zu können, benötigt dieses eine bestimmte Software. Am Parkplatz angekommen, fährt das Fahrzeug in das Sichtfeld des Parkautomaten, auch *Registrierungszone* genannt (1). Wird ein Fahrzeug erkannt, registriert sich der Parkautomat beim Fahrzeug. Ist die Registrierung im Fahrzeug angekommen überprüft der Software Applikation Manager, ob die für den Service benötigte Software bereits auf dem Fahrzeug installiert (2).

Ist dies der Fall, wird zusätzlich überprüft ob die installierte Version der Software aktuell ist. Wenn ja, kann der Service genutzt werden - wie die Nutzung eines Service abläuft, wird im dritten und letzten Kommunikationsprotokoll dargestellt. Ist die Software noch nicht installiert oder die installierte Version ist nicht mehr aktuell wird der Fahrzeughalter gefragt, ob er die Software aktualisieren bzw. kaufen möchte (4). Er kann das ihm vorgestellte Angebot entsprechend seiner Bedürfnisse für die Software anpassen (*Leihe statt Kauf, etc.*) und seine Entscheidung durch eine Eingabe in der Mensch Maschine Schnittstelle bestätigen. Wird die Anfrage abgelehnt, wird keine Software installiert. Das Fahrzeug verlässt die Registrierungszone wieder und kann nicht auf dem Parkplatz parken. Nimmt der Fahrzeughalter die Anfrage hingegen an, sendet das Fahrzeug eine Installationsanfrage an den Server. Der Server nimmt die Anfrage an und erstellt ein Installationspaket für die entsprechende Software (5). Das erstellte Paket wird an das Fahrzeug gesendet, auf diesem installiert und das Fahrzeug kann den angebotenen Park-Service nutzen.

4.3.3 Kommunikationsprotokoll: Nutzung eines Service

Abbildung 12 zeigt den Kommunikationsablauf zwischen einem Fahrzeug und einem Service Provider Actor (*bspw. ein Parkautomat*). Der dargestellte Ablauf findet statt, wenn ein Fahrzeug einen Service nutzen möchte und die aktuelle Version der hierfür notwendigen Software bereits auf dem Fahrzeug installiert ist (1). Der Service wird über die Mensch Maschine Schnittstelle zur Nutzung vorgeschlagen (2) und der Fahrer kann entscheiden, ob er dies tun möchte oder nicht. Die Entscheidung des Fahrers wird an den Parkautomaten gesendet (3), welcher die Entscheidung evaluiert und dementsprechend dem Fahrzeug entweder einen Parkplatz zuweist oder es verabschiedet. Die jeweilige Antwort wird an das Fahrzeug gesendet und über die MMS angezeigt. Das Fahrzeug plant die kommende Bewegung und fährt dementsprechend entweder zu dem ihm zugewiesenen Parkplatz oder verlässt die Registrierungszone wieder.

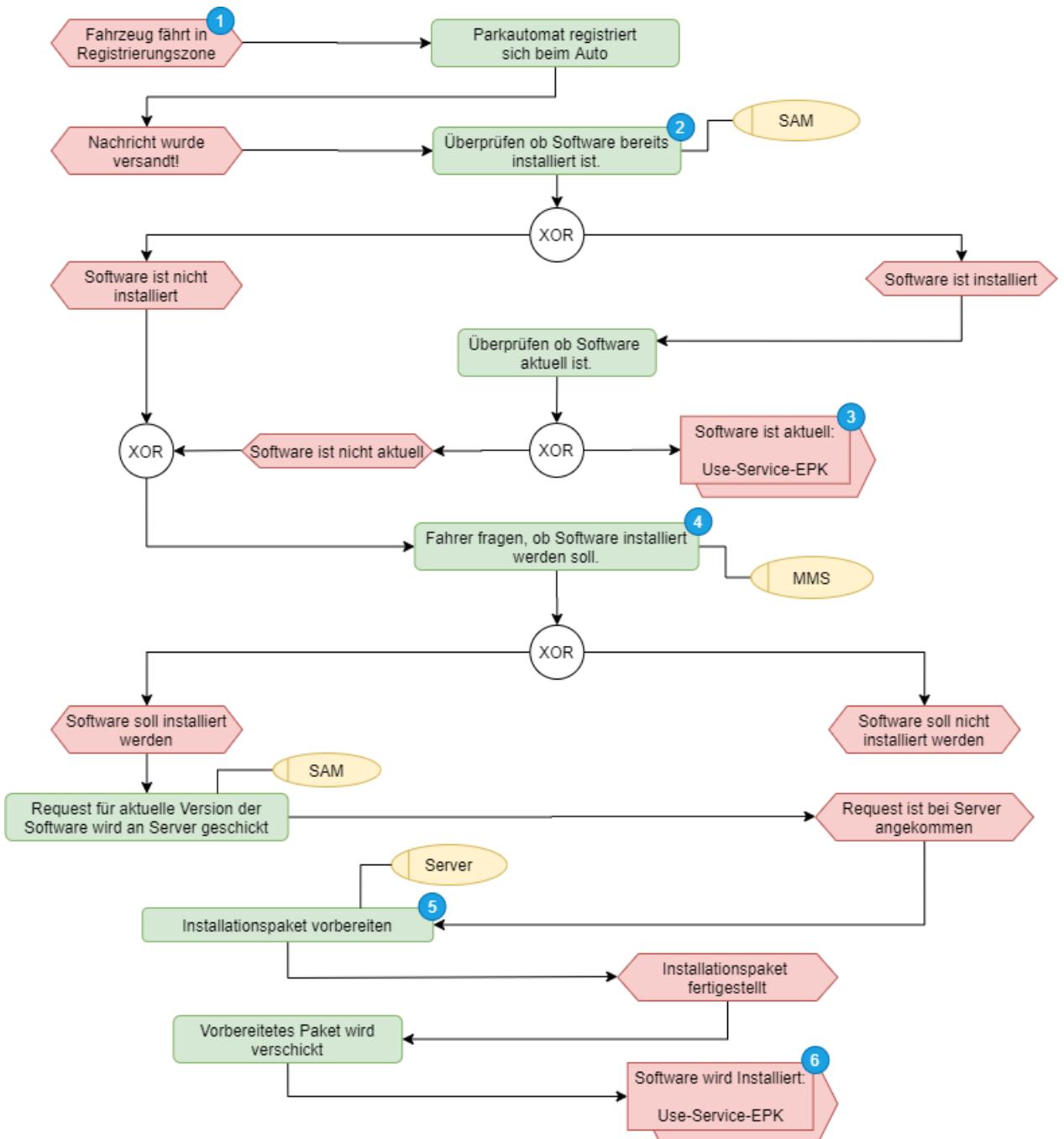


Abbildung 11: Installationsprozess von Software

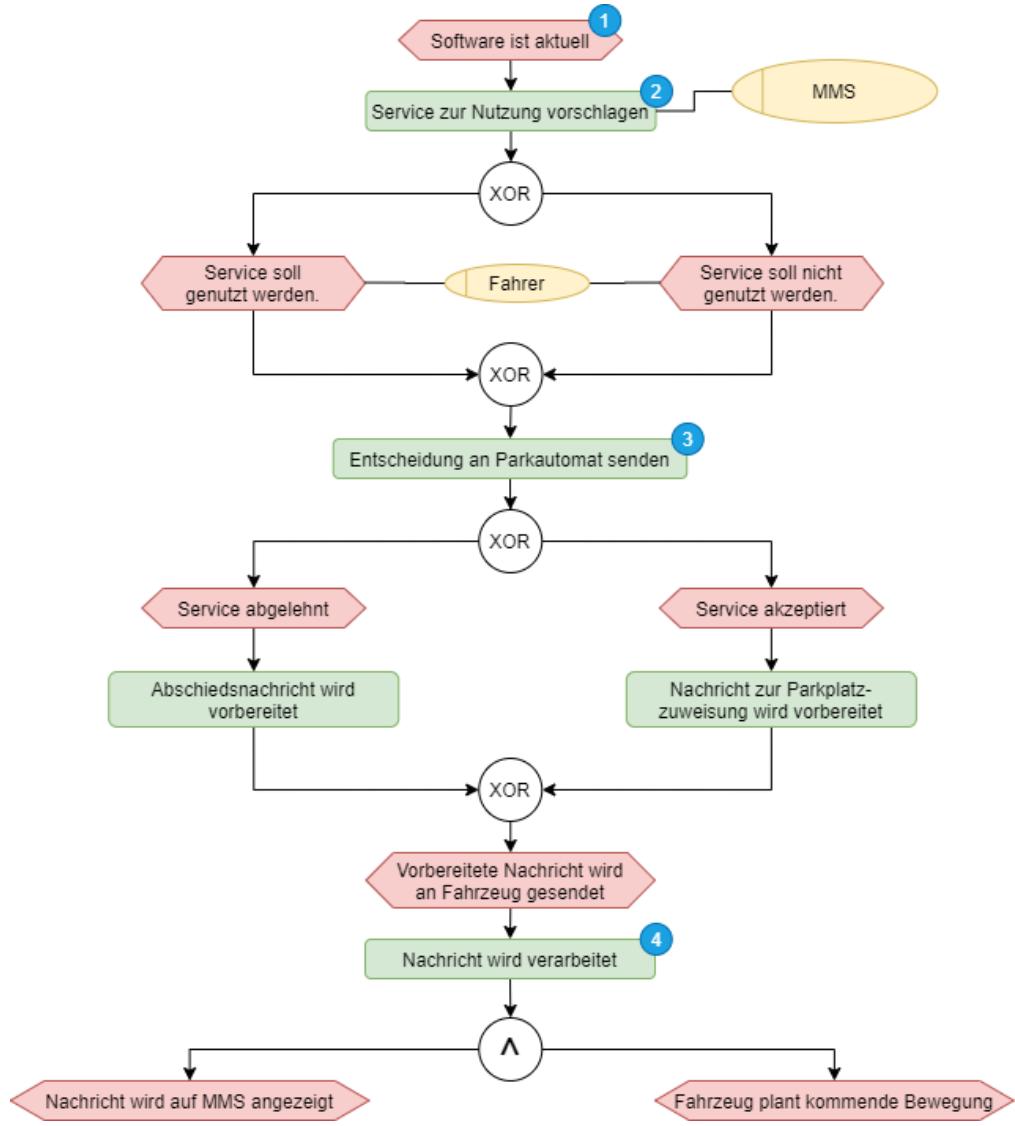


Abbildung 12: Nutzungsprozess von Software

4.4 Ausblick

Die in diesem Kapitel erstellten Konzepte wurden mit der Intention entwickelt, einzelne Bausteine der Wertschöpfungskette zu detaillieren und im Prototypen implementiert zu werden. Die Identifikation von Berechnungen für einzelne Systeme eines Fahrzeugs bieten eine geeignete Grundlage, die Sicherheit intelligenter Fahrzeuge zu unterstützen. Fahrzeughalter können durch diese und weitere Kennzahlen bessere Kaufentscheidungen treffen und ihr Fahrzeug so erweitern, dass es auf deren Ansprüche angepasst ist. Die Kommunikationsprotokolle wurden erstellt, um einen Ablauf für den Prototypen aufzustellen und nachzuverfolgen zu können. Die Kommunikationsprotokolle, die im Forschungsseminar erarbeitete Softwarearchitektur und auch weitere in der Arbeit gewonnene Erkenntnisse und Ideen werden im Prototypen eingebunden und visuell veranschaulicht.

5 Vorstellung des Prototypen

Der Prototyp demonstriert, wie eine Software entsprechend der in Kapitel 3.1 identifizierten Bausteine einem Fahrzeughalter bereitgestellt werden kann. Konkret wird die Bereitstellung einer Software dargestellt, mit welcher **ein Fahrzeug selbstständig ein Parkticket kaufen und anschließend zum Parkplatz fahren kann**. Zunächst wird in Kapitel 5.1 dargestellt, wie der Prototyp installiert werden kann. Im Anschluss daran wird in Kapitel 5.2 die Software Architektur des Prototypen vorgestellt. Abschließend werden die Funktionen des Prototypen vorgestellt und wie diese genutzt werden können.

5.1 Architektur des Prototypen

Abbildung 13 zeigt die Grundlegende Architektur des Prototypen. Über den OEM-Server (*Server*) können Softwares heruntergeladen werden. Der **Software Applikation Manager (SAM)**, die **Carla Simulation** und die **Mensch Maschine Schnittstelle (MMS)** bilden gemeinsam die Repräsentation des Fahrzeugs. Die Python-basierte Carla Simulation stellt das Fahrzeug in einer Stadt dar. Dieses Fahrzeug ist die visuelle Repräsentation des Prototypen. Dies restlichen Module sind alle in Java geschrieben. Die Mensch Maschine Schnittstelle ist eine Android-Schnittstelle, auf welcher eine Version des Software Shops als App installiert ist. Über diese App können Nachrichten von dem Software Applikation Manager empfangen und verarbeitet werden. Außerdem können Nutzereingaben getätigt werden um mit dem restlichen Komponenten des Fahrzeugs

zu kommunizieren. Der Software Applikation Manager ist mit allen Modulen des Prototypen verbunden und stellt daher die zentrale Steuerungseinheit des Prototypen dar. Außerdem erfolgt die Installation und Nutzung von Software auf diesen.

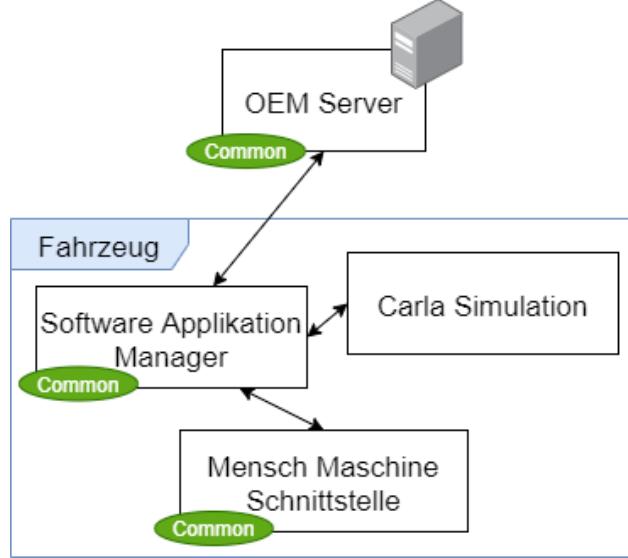


Abbildung 13: Grundlegende Architektur der Prototypen

Der Server, SAM und die MMS kommunizieren alle auf Basis der **Common-Objekte**. Diese sind eine Sammlung an Java-Objekten, welche für die Zweckgebundene Kommunikation jedem einzelnen Modul bekannt sein muss. Sie enthalten Primitive Werte (*Integer, String, Long, etc.*) und leiten alle vom Serializable-Interface ab. Abbildung 14 stellt die generelle Aufteilung der Objekte in zwei Gruppen dar.

Environment-Objekte sind Objekte, welche zur Umwelt des Prototypen gehören. Damit der Server spezifische Softwares verschicken kann, wird ihm die abstrakte Klasse Software zur Verfügung gestellt als auch eine Sammlung von Softwares. Jede Software des Prototypen benötigt eine eindeutige ID und eine Methode *handleMessage()*, welche bestimmte Messages bearbeiten kann. Auch der Software Applikation Manager nutzt diese Objekte, um so die Softwares *installieren und nutzen* zu können. Die Description wird genutzt, um Softwares oder Services, die im Prototypen implementiert werden zu beschreiben. Das Provider-Objekt tut eben dies für die Beschreibung des Anbieters der Software bzw. des Services. Das Angebot repräsentiert die Auswahlmöglichkeiten, die dem Fahrzeughalter beim Kauf einer Software über die MMS vorgeschlagen werden (*Kauf, Leih, Abo*). Alle Drei Objekte werden in der MMS genutzt um das Angebot darzustellen.

Neben den Environment-Objekten gibt es auch noch Messages. Messages sind die Objekte, die im Prototypen zwischen den einzelnen Komponenten verschickt werden. Sie beinhalten neben Primitiven Datentypen auch Environment Objekte und sind ebenfalls für die zweckgebundene Kommunikation gedacht. Messages werden in SoftwareMessages oder ServiceMessages unterteilt. SoftwareMessages sind all jene Nachrichten, welche zur Installation oder Nutzung einer Software notwendig sind und ServiceMessages sind jene, die zur Nutzung eines Service notwendig sind.

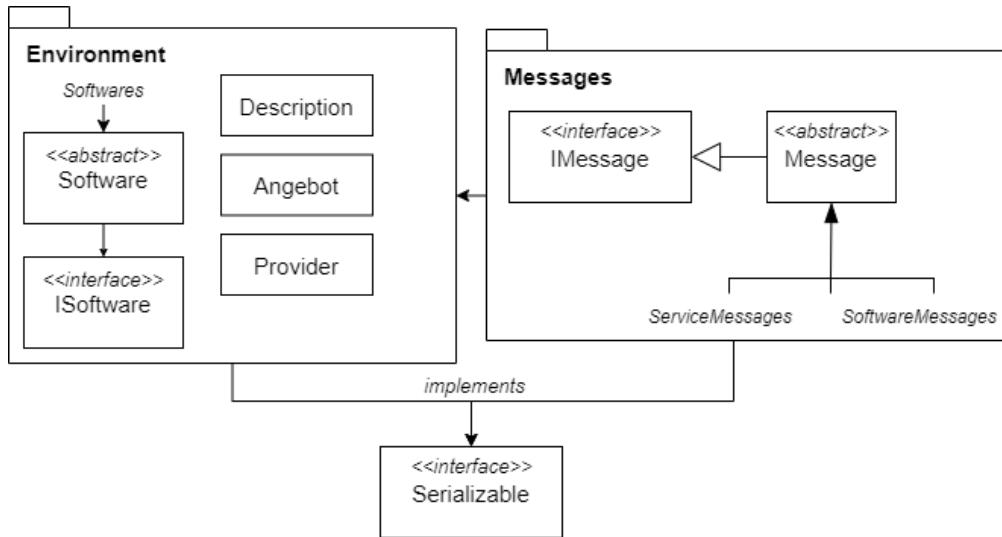


Abbildung 14: Aufbau der Common Library

5.1.1 Architektur des Software Applikation Managers (SAM)

Wie zuvor erwähnt, wird über den Software Applikation Manager das Geschehen des Prototypen gesteuert. Er ist, wie in Abbildung 15 zu sehen ist, in drei kleine Module aufgeteilt. In der GUI werden alle Ereignisse des Prototypen in einem Log dargestellt. Die Log-Einträge werden vom MessageHandler eingetragen. In der Software Control Unit baut der MessageHandler Netzwerkverbindungen zwischen sich, dem Server, der MMS und Carla auf. Jede eingehende Nachricht wird von ihm empfangen und verarbeitet. In diesem Kontext, kann eine eingegangene Nachricht an das MMS weitergeleitet werden oder es wird auf eine Nutzereingabe in der GUI gewartet. Außerdem beinhaltet der Messagehandler den SoftwareManager. Dieser verwaltet die auf dem "Fahrzeuginstallierten Softwares und stellt diese zur Verfügung wenn sie zur Auswertung einer Nachricht benötigt werden.

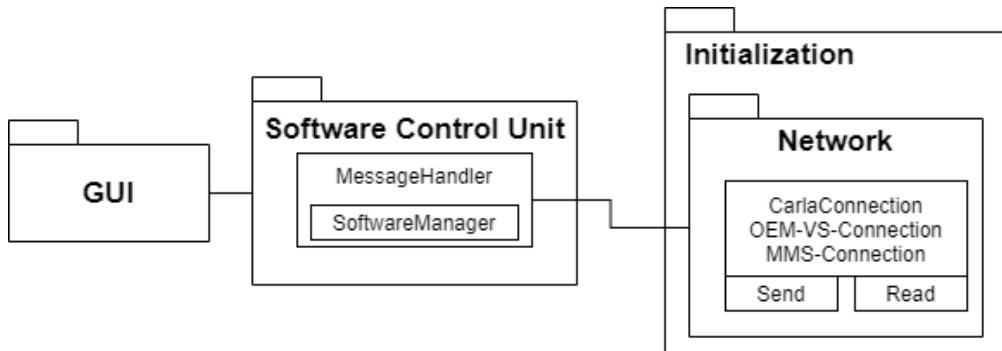


Abbildung 15: Architektur des Software Applikation Managers

5.1.2 Architektur des OEM-Servers (Server)

Die Architektur und auch die Funktionalität des Servers wurden unter der Beachtung des Uptane Standards geplant und implementiert. Die Architektur ist in Abbildung 16 dargestellt. Der Kern des Servers ist der **Director**, welcher eingehende Anfragen für Softwares abarbeitet. Der Director ist mit dem **Image Repository** verbunden, in welchem sich die Softwares befinden, welche im Shop bereits bereitgestellt werden. Uptane gibt vor, eine **Inventory Database** zu führen, in welcher die Manifeste aller im Shop registrierter Fahrzeuge in der Form gespeichert sind, in welcher sie zuletzt von dem Director verifiziert wurden. Wird eine Anfrage an den Server geschickt, enthält diese das aktuelle Manifest des Fahrzeugs. Der Director vergleicht dieses mit dem Manifest, welches in der Inventory Database gespeichert ist. Durch den Vergleich des ByteCodes beider kann festgestellt werden, ob am Fahrzeug Änderungen von einer externen Instanz getätigt wurden oder nicht.

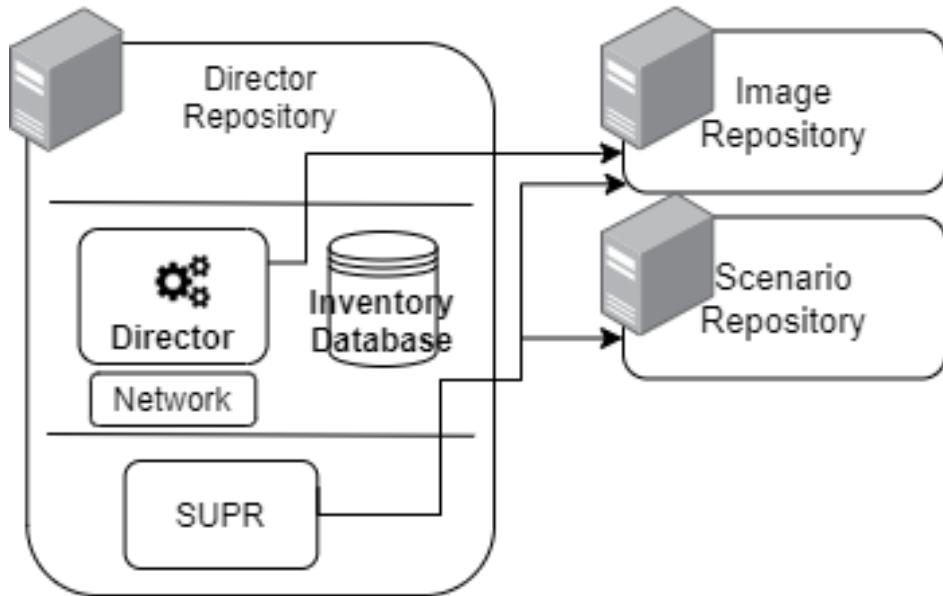


Abbildung 16: Architektur des Software Applikation Managers

Der Director kann eingehende Anfragen an den Software-User-Pattern-Recognizer (*SUPR*) weiterleiten, welcher in Kapitel zwei vorgestellt wurde. Der SUPR ist dafür zuständig, den Software-Bedarf von Fahrzeughaltern anhand ihrer Fahrdaten zu identifizieren. Er hat Zugriff auf das Image Repository und das *Scenario Repository*. In diesem werden, wie in Kapitel zwei erwähnt, die zu einer Software hinzugefügten Szenarien gespeichert. Der Director leitet Anfragen weiter, die das feststellen von Softwarebedarf umfassen. Er selber verarbeitet nur die direkten Kaufanfragen für Software.

5.2 Installation

Die Installation der einzelnen Module muss getrennt voneinander passieren. Für die vollständige Nutzung muss sichergestellt sein, dass Carla auf dem PC auf welchem der Prototyp installiert werden soll lauffähig ist. Schauen Sie sich hierzu die Systemanforderungen von Carla an.¹⁸ Die einzelnen Module können auch auf

¹⁸https://carla.readthedocs.io/en/latest/start_quickstart, Aufgerufen am 14. Juni 2020

getrennten Geräten genutzt werden, indem die IP-Adressen und Ports der Module über deren Config-Dateien angepasst werden. Die folgende Anleitung stellt dar, welche Schritte zur Installation der einzelnen Module getätigt werden müssen. Sie ist für Windowsgeräte ausgelegt und dauert ca. 30-40 Minuten.

5.2.1 Server und SAM Installation

1. .zip-Ordner Herunterladen

Gehen Sie auf <https://github.com/hesty98/bachelorthesis>. Klicken Sie auf den "releases"-Tab, welcher unter der Beschreibung des Repository zu finden ist. Laden Sie die Datei "*prototyp.zip*" herunter und entpacken Sie den Ordner auf Ihrem Laptop. In diesem Ordner finden Sie die .jar-Dateien mit welchen der Server und der SAM gestartet werden können. Der Ordner "*PythonScriptsThesis*" wird später für die Installation von Carla benötigt.

2. IP-Adresse aufschreiben

Um das MMS und auch Carla mit dem Software Applikation Manager zu verbinden, müssen Sie Ihre IP-Adresse kennen. Öffnen Sie hierzu Ihr Windows-Temrinal und geben Sie *ipconfig* ein. In der Zeile IPv4-Adresse steht Ihnen die IP-Adresse. Diese müssen Sie sich merken.

5.2.2 MMS Installation

1. Android Studio herunterladen

Sie haben zwei Optionen, wie die Mensch Maschine Schnittstelle genutzt werden kann. Entweder sie simulieren das Android Gerät auf Ihrem PC oder Sie installieren die App auf ein eigenes Tablet/Smartphone. Für beides wird eine aktuelle Version von Android Studio benötigt, die Sie hier <https://developer.android.com/studio> herunterladen können. Installieren Sie Android Studio und fahren Sie anschließend vor.

2. Code importieren

Das Repository der Mensch Maschine Schnittstelle liegt auf GitHub. Über Android Studio können Sie das Projekt direkt importieren. Klicken Sie hierfür auf "File > new > Project from Version control > GitHub" und geben Sie folgende URL ein: <https://github.com/hesty98/HMI>.

TESTEN

Alternativ kann das Projekt auch als .zip-Datei heruntergeladen werden. Der extrahierte Ordner kann in Android Studio als Projekt geöffnet werden (*File -> open -> Ordner wählen*).

3. MMS Konfigurieren

Öffnen Sie in Android Studio die Datei Connection.java und passen Sie den Host entsprechend der IP des *Software Applikation Managers* an. Die Datei befindet sich im 'network'-Ordner (*app -> java -> com.linushestermeyer.hmi -> network*).

4. MMS aufsetzen

Sie können die Mensch Maschine Schnittstelle nun entweder auf einem eigenen Gerät installieren

oder das Gerät auf Ihrem PC simulieren. Für beide Optionen Sie zuerst eine Konfiguration für die App erstellen. Folgen Sie hierzu folgender Anleitung: <https://developer.android.com/studio/run/rundebugconfig>.

4.1 Eigenes Tablet

Zur Installation auf Ihrem Tablet müssen Sie zunächst die *Entwickler-Optionen* für dieses Freischalten und *USB-Debugging* aktivieren. Folgen sie hierfür folgender Anleitung: <https://mobil-sicher.de/ratgeber/usb-debugging-aktivieren>. Wurde dies getan, kann das Gerät an den PC angeschlossen werden. Über den grünen 'Play'-Button in der oberen Menuleiste kann die App auf Ihrem Gerät installiert werden.

4.2 Virtual Device

Wenn Sie Die App **nicht** auf einem eigenen Gerät installieren wollen, müssen sie mit Android Studio ein Simulierte Gerät erstellen. Dies funktioniert nur, wenn ihr PC eine **Intel-CPUs** hat, da Android Studio den *HAXM-Service* von Intel zur Simulation nutzt. Verfügt ihr PC über eine Intel-CPUD, können Sie mit Hilfe folgender Anleitung ein Virtuelles Gerät erstelle: <https://developer.android.com/studio/run/managing-avds>. Erstellen Sie eine Instanz eines Tablets, nicht die eines Smartphone.

Haben Sie ein *VD* erstellt, können sie anschließend über den grünen 'Play'-Button in der oberen Menuleiste die App auf diesem Gerät installieren.

5.2.3 Carla Installation

1. Python installieren und SYS_PATH hinzufügen

Damit Carla auf dem PC laufen kann, muss Python in Version 3.7 auf dem PC installiert sein und dem SYSTEM_PATH hinzugefügt werden. Hierzu kann folgende Anleitung genutzt werden: <https://geek-university.com/python/add-python-to-the-windows-path/>. Hier die 3.4 nur durch eine 3.7 gedanklich ersetzen. Der PATH kann by der Installation auch durch das ankreuzen einer Checkbox automatisiert werden.

2. Carla Herunterladen

Carla kann als .zip-Ordner heruntergeladen werden. Die Dateien müssen in einen eigenen Ordner auf dem PC extrahiert werden. Als nächstes öffnen sie den Speicherort von Carla im Explorer und öffnen in diesem Fenster das Terminal (*geben sie 'cmd' in den Suchbalken des Explorers ein*). Jetzt müssen die für die Simulation notwendigen Python-Packages *pip*, *pygame* und *numpy* installiert werden. Die pip-Installation kann hier nachgelesen werden: <https://pypi.org/project/pip/>. Für die Installation von pip sollte zudem eine Adminsitrator-Konsole verwendet werden (*Windows -> suche cmd' -> 'Als Administrator öffnen'*). Ist pip installiert, können mittels folgender Zeile die übrigen packages hinzugefügt werden:

```
py -3.7 -u pip install --user pygame numpy  
EINFÜGEN HILFE
```

Im Anschluss daran sollten Sie Carla ein mal testen. Starten Sie zunächst die Carla.exe und wechseln sie im Terminall in den PythonAPI/examples Ordner (*cd PythonAPI/examples*). Führen Sie anschließend folgenden Befehl durch:

```
py- 3.7 spawn_npc.py
```

3. Skripte einfügen

In der extrahierten .zip-Datei des SAM und Servers befindet sich ein Ordner mit dem Namen *PythonScriptsThesis*. Kopieren Sie diesen Ordner in den *PythonAPI*-Ordner von Carla und Sie haben es geschafft!

4. Carla Konfigurieren

Abschließend sollten sie noch Konfigurationen an Carla vornehmen. Zuerst wird die Stadt geändert in welcher sich das Szenario abspielen wird. Hierzu

EINFÜGEN

Haben Sie die entsprechenden Werte eingetragen, speichern und schließen Sie die Datei. Anschließend wechseln sie in den Ordner *PythonAPI/PythonScriptsThesis* und editieren Sie die `install_sw_use_service.py`. Passen Sie die IP an die des **Software Applikation Managers** an.

5.3 Nutzung des Prototypen

Die Module des Prototypen müssen in bestimmter Reihenfolge gestartet werden:

1. Server

Server.jar ausführen.

2. Software Applikation Manager

SAM.jar ausführen

3. Carla

Starten Sie Carla. Führen Sie anschließend im Terminal im Ordner "*PythonAPI/PythonScriptsThesis*" das Skript `install_sw_use_service.py` aus.

```
py -3.7 install_sw_use_service.py
```

4. Mensch Maschine Schnittstelle

Starten Sie die App auf ihrem Gerät oder in dem Virtual Device. Gehen Sie sicher, dass die App zuvor geschlossen war.

Im folgenden haben Sie fünf Bildschirme, über die alle Module des Prototypen sichtbar sind. Die Mensch Maschine Schnittstelle wartet auf eingehende Nachrichten, ebenfalls wie Carla. Die GUI des Software Applikation Managers ist in drei Bereiche aufgeteilt. Der Log auf Linken Seite stellt alle Ereignisse aus Sicht des Fahrzeugs dar. Der Log auf der Rechten Seite der GUI stellt sämtliche Ereignisse dar, die von dem Parkautomaten wahrgenommen werden. Zwischen beiden Logs sind diverse Knöpfe, die für die Steuerung des Prototypen gedacht sind.

Auto spawnen spawnt das Fahrzeug in der Carla Simulation. Mit den WASD-Tasten und der Maus können Sie herauszoomen und das Fahrzeug suchen. Prinzipiell ist es links vor ihnen. Der Zum Parkplatz fahren-Knopf lässt das Fahrzeug in der Simulation zu einem Parkplatz fahren und es hält dort an.

Der In Erkennungsbereich fahren-Knopf lässt das Fahrzeug in den Bereich fahren, in welchem der Parkautomat dieses erkennen kann. Nun kann sich der Parkautomat über den Sende Service Registrierung-Knopf beim Fahrzeug registrieren. Der Sende ActionMessage-Knopf sendet eine Nachricht an Carla, die entweder das Auto parken lässt oder es vom Parkplatz entfernen lässt. Das Szenario kann durch den Neustarten-Knopf beliebig oft durchgeführt werden. Durch den Hack Car!-Knopf wird das Auto "gehackt". Dies soll den Fall darstellen, wenn ein Fahrzeug versucht eine Software vom Shop zu installieren aber extern Änderungen im Auto vorgenommen wurden. Durch Änderungen am Fahrzeug verändert sich das Manifest des Fahrzeug, weshalb ein Fahrzeug keine Kommunikation mit dem Server aufbauen kann. Sämtliche Ereignisse die in Folge eines Knopfdrucks passieren, werden in den Logs notiert.

Über die **Mensch Maschine Schnittstelle** kann auf vom SAM verschickte Nachrichten reagiert werden. Initial ist der Home Screen des Shops zu sehen. Erhält das die MMS eine Software- oder ServiceRegistration-MESSAGE, erscheint ein PopUp, in welchen der entsprechende Service bzw. die Software vorgestellt wird. Es kann entschieden werden, welches Angebot genommen werden soll oder auch die komplette Transaktion mit **Cancel** abgebrochen werden.

Die Server-GUI und die Simulation stellen weitere Visuelle Repräsentationen des Prototypen dar. In beiden können die Auswirkungen nachvollzogen werden, die durch die MMS und den SAM geschehen.

5.4 Analyse des Prototypen und Ausblick auf die Weiterentwicklung

Im Prototypen wurde die im Forschungsseminar erläuterte Uptane Architektur versucht grundlegend zu implementieren. Hierdurch konnte gezeigt werden, wie ein Fahrzeug in der Zukunft von externen Eingriffen geschützt werden könnte. Die erstellte Software Architektur lässt eine einfache Erweiterung des Prototypen zu. Für einen neuen Anwendungsfall muss bloß eine neue Software erstellt werden und das Carla Skript muss angepasst werden. Dies dauert ca. zwei Stunden. Die Installation des Prototypen ist relativ Zeitaufwendig und komplex, weshalb das Bereitstellen einer detaillierte Anleitung wichtig gewesen ist.

Der Prototyp hat dargestellt, wie eine Software über eine kabellose Schnittstelle ort- und zeitunabhängig gekauft und installiert werden kann. Zusätzlich wurde die Interaktion mit den in der Wertschöpfungskette identifizierten Service Providern implementiert, was zusätzliche Anwendungsfälle bieten kann. Über die GUIs kann nachvollzogen werden, was aktuell im Fahrzeug passiert und die Simulation kann ausreichend gesteuert werden. Die visuelle Repräsentation des Fahrzeugs in Carla unterstützt die Wirkung des Prototypen enorm, da der Betrachter direkt die Auswirkungen seiner Entscheidungen sieht. Auch die Interaktion über die Android-basierte Mensch Maschine Schnittstelle hat sich als sinnvoll erwiesen, da hierdurch ein guter Bezug zur Realität und Android Automotive gezogen werden kann.

Der Prototyp bietet noch eine Vielzahl an Erweiterungsmöglichkeiten. Der **Server** könnte um weitere Uptane-Funktionen erweitert werden. Die Erstellung eines Server-seitigen SUPR kann Türen öffnen, komplexe Suchalgorithmen auf der Basis von OpenScenario-Dateien in den Prototypen zu integrieren. Auch der **Software Applikation Manager** kann durch diverse neue Funktionen Sinnvoll erweitert werden. Ein Fahrzeug-basierter SUPR, dargestellt in Kapitel 2.3.3, kann Bestimmen von ein PopUp auf der MMS erscheinen soll und sodurch das wahllose erscheinen von PopUps verhindern. Die Steuerung über die GUI sollte

den neuen Anwendungsfällen entsprechend angepasst werden, da nicht in jedem Anwendungsfall eine Kommunikation mit einem Service Provider stattfindet. Entfernt man sämtliche Nutzereingaben des Prototypen und automatisiert so die gesamte Simulation, können die dargestellten Anwendungsfälle beliebig hoch skaliert werden, wodurch ein Shop und die Zuverlässigkeit der Bereitstellung getestet werden kann.

Im Laufe der Entwicklung wurde deutlich, dass Carla äußerst umfangreich ist. Würden die Funktionen von Carla ausgereizt werden, könnte eine Vielzahl neuer Anwendungsfälle zum Prototypen hinzugefügt und in Carla visuell dargestellt werden. Auch eine Wechsel zwischen autonomen Fahrfunktionen und eigenständiger Steuerung des Fahrzeugs kann in Carla erfolgen. Im aktuellen Prototypen ist Carla nur eine visuelle Repräsentation. Durch eine Implementierung des SAM in Python kann dies korrigiert werden und die Architektur kann um ein Modul verkleinert werden. Carla wäre dadurch sowohl die visuelle Repräsentation des Fahrzeugs als auch die "Business Logik".

6 Reflexion der Ergebnisse

Die zu Beginn der Bachelorarbeit aufgestellte Forschungsfrage stellte die Anforderung, wichtige Bausteine einer Wertschöpfungskette für die Erkennung und Bereitstellung neuer Fahrfunktionen zu identifizieren und darzustellen, wie diese von Automobilherstellern umgesetzt werden können. Das erstellte Business Model Canvas wurde mit der Intention erstellt, als Einführung in den Markt für Fahrfunktionssoftware zu dienen. Die hierdurch zusätzlich entstandenen Inhalte haben eine sehr gute Grundlage für einen "Blick über den Tellerrand" geboten, welcher eine mögliche Zukunft von Fahrzeugen veranschaulicht.

Die in der Wertschöpfungskette identifizierten Bausteine haben viele Aufgaben eines Software Shops aufgedeckt. Einige der Bausteine wurden in dieser Arbeit aus Platzgründen nicht im Detail erläutert, damit die im Fokus stehenden Bausteine der Bedarfserkennung und Bereitstellung ausreichend detailliert werden konnten. In den Kapiteln 2 und 4 wurden technischen Konzepte erstellt um diese Bausteine tiefer gehend zu betrachten. Die Ergebnisse dessen sind zufriedenstellend, da sie einen Überblick über die verschiedenen Aufgabenfelder der Erkennung und Bereitstellung von Software bieten. Dennoch hätten einige dieser Bausteine qualitativ besser erörtert werden können, zum Beispiel durch die Integration von mehr Bausteinen in den Prototypen. Dieser hat letzten Endes die vier folgenden Bausteine der Wertschöpfungskette visualisiert:

- Betreiben des Shops
- Automatische Erkennung von Softwarebedarf
- Angebotsunterbreitung
- (Sicherer) Download über das Internet

Durch die entwickelte Architektur können diesem jedoch weitere Bausteine und neue Anwendungsfälle mit nur geringem Zeitaufwand hinzugefügt werden. Die GUI des Fahrzeugs stellt die dort geschehenden Ereignisse gut dar und die Steuerung über Knöpfe ist zielführend, jedoch nur für den Anwendungsfall der Bachelorarbeit ausgelegt. Änderungen in der Architektur wie das Auslagern des Anwendungsfalls hätten die Erweiterung zusätzlich vereinfacht. Die Integration des Uptane-Standards in den Prototypen und die Einbindung von OpenScenario-Dateien in technische Konzepte hat die Wichtigkeit von OpenSource-Lösungen verdeutlicht für Fahrzeuge aufgezeigt.

Einige Ergebnisse wie zum Beispiel die Bausteine des Supply Chain Managements oder des generellen Betriebs haben keinen direkten Bezug auf die Forschungsfrage der Bachelorarbeit. Diese Problematik entstand durch eine Unterschätzung des Themas zu Beginn. Es wurde erst im Laufe der Erarbeitung deutlich, dass zum Verständnis einiger Bausteine der Bedarfserkennung und Bereitstellung weiteres Grundwissen notwendig ist, welches durch das Business Model Canvas vermittelt wurde. Die Integration dieser zusätzlichen Informationen hat letzten Endes zu einem besseren Gesamtergebnis beigetragen und viele Blickwinkel eröffnet wodurch mehrere Themen für Abschluss- und Forschungsarbeiten durch diese Arbeit vorgeschlagen werden können.

7 Ausblick

In direkter Anknüpfung an die Arbeit sollten die identifizierten Bausteine weiter erforscht und im Prototypen veranschaulicht werden. Es sollte eine GUI für den Server erstellt werden, damit auch die dortigen Entscheidungen nachvollzogen werden können. Außerdem kann die Simulation in Carla durch weitere Verkehrsteilnehmer realistischer gestaltet werden. Durch die Implementierung des "Software User Pattern Recognizer" (SUPR, Kapitel 2) des Fahrzeugs und des Servers würde den Prototypen vervollständigen. Ein Prototyp, der diese Anforderungen erfüllt und zudem mehrere Anwendungsfälle umfasst, kann sehr gut für die Vorstellung auf Messen oder anderweitigen Präsentationen genutzt werden, um "die Zukunft des Autofahrens" zu zeigen.

In der Arbeit wurde davon ausgegangen, dass Software auf einem einzelnen Fahrzeug installiert wird. Es ist vorstellbar, dass gekaufte Softwares nicht nur auf einem einzigen Fahrzeug genutzt werden sollen wozu vor allem der wachsende Car-Sharing Markt beiträgt. Damit Fahrer sich einfach auf einem einloggen können und im folgenden alle jemals von ihm gekauften Softwares verfügbar sind ist ein technisches Konzept zu erstellen. Dieses soll darstellen, wie Software nicht Fahrzeug-gebunden sondern Fahrzeughalter-gebunden gekauft und genutzt werden kann.

Damit Softwares auf Fahrzeugen verschiedener, möglicherweise konkurrierender Automarken installiert und genutzt werden können muss eine Software-API konzipiert werden. Hierbei besteht die Schwierigkeit, dass selbst Fahrzeuge der selben Automarke unterschiedliche Software- und Hardware-Architekturen aufweisen und eine generalisierte Ansteuerung dieser Komponenten dementsprechend umfangreich gestalten kann.

Literatur

- [1] (Aufgerufen am 10. Januar 2020) Andrea Rosenbusch. User-centered design in sieben punkten kurz erklärt. https://zeix.com/wp-content/uploads/2011/04/ict-jb2011_artikel_zeix_rosenbusch_felix_final.pdf, 2011.
- [2] (Aufgerufen am 10. Januar 2020) Divya Krishnan. Ui/ux for autonomous vehicle interface to build trust. <https://medium.com/divya-krishnan-design/ui-ux-for-autonomous-vehicle-interface-to-build-trust-de7f4c545c3b>, 2019.
- [3] (Aufgerufen am 10. Januar 2020) GrummelJS. Datei:walter e. boomer.jpg. https://de.wikipedia.org/wiki/Datei:Walter_E._Boomer.jpg, 2008.
- [4] (Aufgerufen am 10. Januar 2020) Hans-Christian Reuss. Intelligente fahrzeuge. https://www.uni-stuttgart.de/presse/archiv/themenheft/07/intelligente_fahrzeuge.pdf, 2010.
- [5] (Aufgerufen am 10. Januar 2020) Jeff Schneider. Ri seminar: Jeff schneider : Self driving cars and ai. https://www.youtube.com/watch?v=jTio_MPQRYc&t=1497s, 2019.
- [6] (Aufgerufen am 10. Januar 2020) Juergen Daunis. The automotive industry in transformation – business model disruption. <https://www.ericsson.com/en/blog/2017/11/the-automotive-industry-in-transformation--business-model-disruption>.

- [7] (Aufgerufen am 10. Januar 2020) Linux Foundation. Uptane design. <https://uptane.github.io/design.html>, 2020.
- [8] (Aufgerufen am 10. Januar 2020) Marius Dupius. Open scenario. <https://www.asam.net/standards/detail/openscenario/>, 2015.
- [9] (Aufgerufen am 10. Januar 2020) Martin Seibert. Personas geben zielgruppen gesichter. <https://blog.seibert-media.net/blog/2008/09/12/personas-geben-zielgruppen-gesichter/>, 2008.
- [10] (Aufgerufen am 10. Januar 2020) Michael Green. Angry granpa. [https://commons.wikimedia.org/wiki/File:Angry_Grandpa_-_2015_\(cropped\).jpg](https://commons.wikimedia.org/wiki/File:Angry_Grandpa_-_2015_(cropped).jpg), 2015.
- [11] (Aufgerufen am 10. Januar 2020) Seobility. User centered design.
- [12] (Aufgerufen am 10. Januar 2020) Thomas Weißgerber. Konzepte und methoden des user centered design. http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/326_weissgerber_UCD.pdf, 2015.
- [13] (Aufgerufen am 10. Januar 2020) Tibchris. File:a beautiful girl in her black dress.jpg. https://commons.wikimedia.org/wiki/File:A_beautiful_girl_in_her_black_dress.jpg, 2010.
- [14] (Aufgerufen am 10. Januar 2020) usability toolkit.de. usability-toolkit.de: Usability für webprojekte. <http://usability-toolkit.de/usability/usability-in-web-projekten/>, 2020.
- [15] William Sims Bainbridge. *Berkshire encyclopedia of human-computer interaction*, volume 1. Berkshire Publishing Group LLC, 2004.
- [16] Frauen und Jugend Bundesministerium für Familie, Senioren. Work life balance, August 2005.
- [17] Verband der Automobilindustrie. Automatisierung - von fahrassistenzsystemen zum automatisierten fahren. *VDA Magazin - Automatisierung*, 2015.
- [18] Rilind Elezaj. Autonomous cars: Safety opportunity or cybersecurity threat?
- [19] Referat Fahrzeugstatistik. 'fachartikel: Halter der fahrzeuge'. Technical report, Krasftfahrt-Bundesamt, 2011.
- [20] Lex Fridman. 'human-centered autonomous vehicle systems: Principles of effective shared autonomy'. Technical report, Massachusetts Institute of Technology (MIT), 2018.
- [21] Torsten Klanitz. Wertschöpfungskette.
- [22] Thomas Künneth. *Android 8 - Das Praxisbuch für Java-Entwickler*.
- [23] Meike Lorenzen. Warum Ältere menschen innovationen ignorieren.
- [24] Manager-Wiki. Wertkette (nach porter).
- [25] Markus Reinke. *30 Minuten Verkaufsprychologie*. Gabel Verlag, 2013.
- [26] R. Pokam S. Langlois S. Debernard, C. Chauvin. Designing human-machine interface for autonomous vehicles. Technical report, International Federation of Automatic Control, 2016.

- [27] Gerd-Inno Spindler. *Kaufverhalten und Kaufentscheidung*, pages 39–48. Springer Fachmedien Wiesbaden, Wiesbaden, 2016.
- [28] Startplatz. Business model canvas. <https://www.startplatz.de/startup-wiki/business-model-canvas/>.
- [29] Niklas Stelter. Entwurf und simulation von monitoren für die evaluierung von updates intelligenter fahrzeuge. Master's thesis, CvO Universität Oldenburg, 2019.
- [30] Marc Zimmer. Eltern in technik-fragen auf kinder angewiesen.